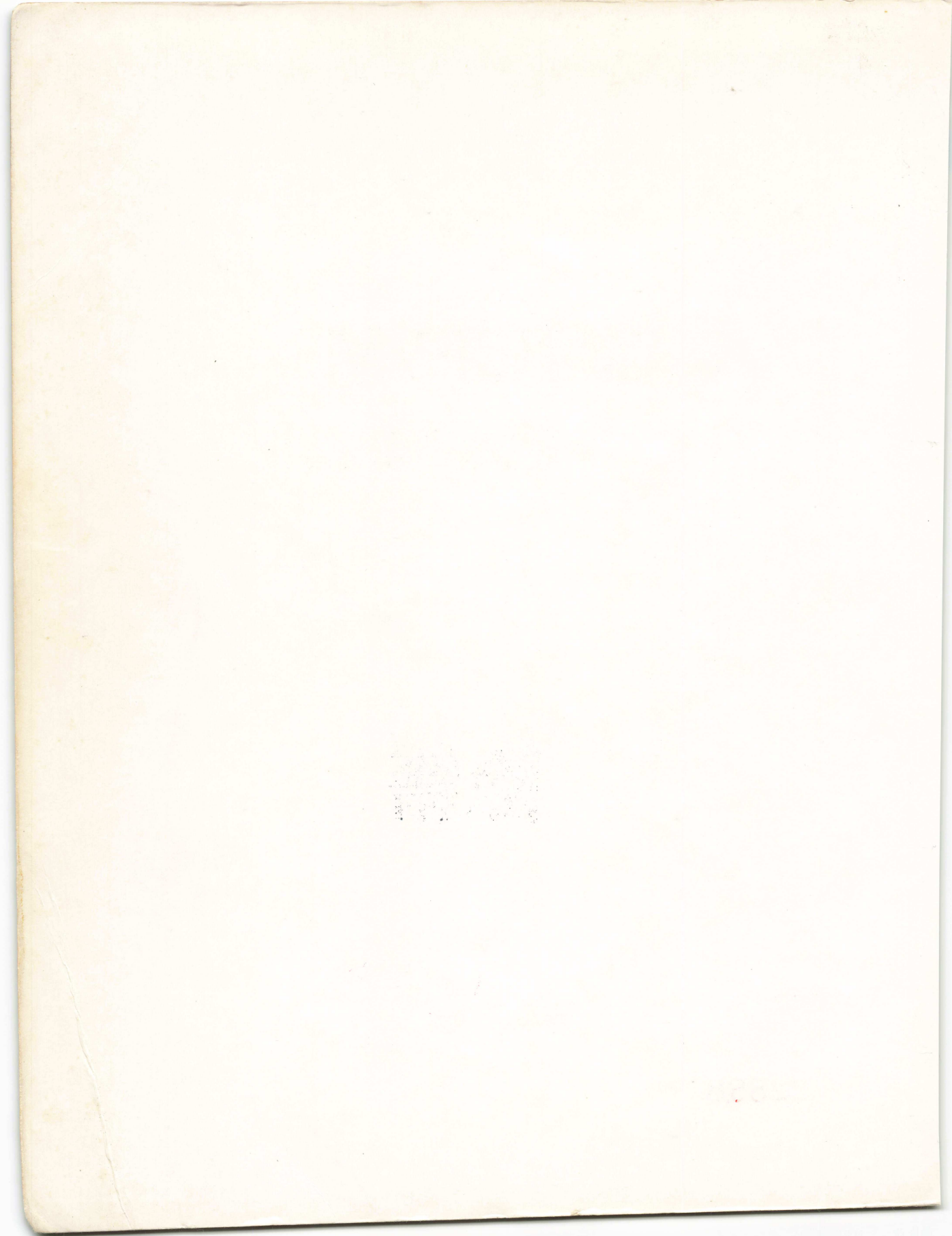


マイクロコンピュータの基本ソフトウェア

応用CP/M®

村瀬康治——著

アスキー出版局

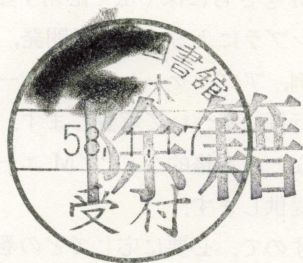


549.92

A
3

応用CP/M®

村瀬 康治 著



アスキー出版局

22888

CP/M Learning System全3巻の構成

この「CP/M Learning System」全3巻は、次のように構成されています。

入門 CP/M……………CP/M がどのようなものであるかを解説し、CP/M を使うための基礎知識、日常よく使う各コマンドの実習などをやさしく具体的に解説します。

今まで CP/M については全く知らなかった読者でも、本書により CP/M の概要を理解し、一通り CP/M が使えるようになるよう配慮されています。

実習 CP/M……………CP/M の全コマンドと、そのほとんどすべての使い方を徹底のかつやさしく、具体的に実習しながら解説します。また、CP/M のハードウェアおよびソフトウェアの構成についても解説し、CP/M の実用例として、CP/M アセンブラによるマシン語開発の全過程を実習します。本書は、読者が本格的に CP/M を使うようになった場合、CP/M のコマンド・ハンドブックとして、随時参照することになるでしょう。

応用 CP/M……………CP/M をさらに深く広く応用する場合についての解説書です。マクロ・アセンブラによるマシン語開発、システム・コール、各種高級言語の使用例、アプリケーションやユーティリティの実行なども、分かり易い実行例に基づいて解説します。さらに CP/M の内部構造、BIOS の詳細など、本格的な CP/M ユーザーとして、いずれは必要となる知識を提供します。

各巻はそれぞれにまとまっていますので、必要に応じてどの巻を手にもされても利用することができます。

本シリーズは、CP/M version 2.2 を基にして書かれていますが、旧バージョンである、version 1.4 を使う場合のことを考慮し、共通でないコマンドについては、そのつど注意書きを付け加えています。

著者まえがき

1982年7月、NECは、ついに自社のパーソナル・コンピュータ、PC-8001とPC-8801に対するCP/Mを、NEC自らが開発し、4万円を切る低価格で発売しました。価格は恐らく国内では今までの最低のものだと思われますが、その性能は、CP/M史上の1つの“事件”として、特記に価する素晴らしいものです。

ディスク・アクセスの高速化、CRT表示の高速化、エスケープ・シーケンスによるスクリーン・コントロール、割り込みを使ったキー入力、N-BASICのROM内ルーチンの呼び出し機能等々、このCP/Mは、今後、他機種用のCP/Mを開発するメーカーにとって、高い所に置かれた1つの目標となるでしょう。パーソナル・コンピュータ用のCP/Mで、このCP/Mを超えるものを作るのは容易なことではないと思われます（このNECのCP/Mについての解説は巻末付録Aを参照して下さい）。また、東芝も同様にPASOPIA用のCP/Mを、本書の原稿が書き上った9月に発売しました。

大手メーカーが、自社の機種用のCP/Mを自ら開発し、発売するに至った背景には、現状のBASICコンピュータのソフトウェア的な行きづまりがあります。

このことは、パーソナル・コンピュータが発売されて以来、付属のBASIC言語によるその応用を考えてきた多くのユーザーが痛感している問題であり、メーカーにとっても、今後の市場を確保し、パーソナル・コンピュータ分野の一層の発展を望むには、ソフトウェアの充実が急務なのです。そのためには8ビット・コンピュータの標準OSとなっているCP/Mを載せなければならないと判断したためでしょう。

今日、「CP/M」は、デジタルリサーチ社という一私企業の製品というより、事実上の世界的標準OSという意味合いが強く、特に8ビット機で80系を採用しているものならば、CP/Mが利用できないパーソナル・コンピュータなどは、メーカーの熱意のなさを示す最たるものと言えるでしょう（学習用機、専用機などは除く）。

「まずCP/Mを」これは筆者が、日本で最初のCP/M解説書である“標準CP/Mハンドブック”以来主張していますが、これは全くユーザーの側に立った発言であり、CP/Mを採用することが最も経済的で、最も多大な利益をもたらすことを否定できる人はいないでしょう。とにかくまず、CP/Mに集中している豊富な共通資源を利用できる体制をとることです。すべてはそれからの話です。もし他のOSに関心があれば、UNIX likeのOSなり、何なりを試みることもよいでしょう。

「まずCP/Mを」これがユーザーの利益にとって最も安全確実な道であると思います。

著者まえがき

本書「応用 CP/M」も、完成が大幅に遅れ、当シリーズの多くの愛読者にしびれをきらさせてしまい、どうも申し訳ありませんでした。それほど怠けていた訳ではないのですが、いざ手を付けてみるとあまりにも多くの事柄があり過ぎ、時間がかかってしまいました。しかし本書の内容を見ていただければ、多少は許してもらえるかな?……とも思います。本書「応用 CP/M」により、初めて世の中に、CP/M の本当の力を紹介できる本が出現したと思っています。

本書の「システム・コール」の章により、多くの読者が、CP/M 最大のメリットであるこのシステム・コールを自在に使いこなせるようになるでしょう。それにより、CP/M 上の多くのアプリケーション・ソフトが生まれてきます。

また「高級言語」の章に刺激を受け、自分の目的に合う言語に取り組む読者も大勢いるでしょう。そこからは、アセンブラで書くのが困難な大きなシステム・ソフトウェアも生まれてきます。

こうして、共通 OS のもとで、優れたソフトウェアが生み出され、それが広く流通し、日本のマイクロコンピュータのソフトウェア水準の向上に、当 CP/M シリーズが少しでも役立つなら、著者としてこれ以上の喜びはありません。そうなることを心から期待しています。

1982年 9 月 村瀬康治

目次 ●

CP/M Learning System 全3巻の構成.....	(2)
著者まえがき.....	(3)

★ 1章 CP/Mの内部構造と機能の詳細 1

1.1 BIOS(基本入出力システム).....	4
1.1.1 BIOSの構成と機能.....	5
1.1.2 ディスク・パラメータ・テーブル.....	15
1.1.3 DISKDEFマクロ・ライブラリの使い方.....	21
1.1.4 セクタ・ブロッキング, デブロッキングの概念.....	30
1.2 BDOS(基本ディスク・オペレーティング・システム).....	34
1.2.1 ファイル・コントロール・ブロック(FCB).....	34
1.2.2 プログラム実行時のコマンド・ラインとFCBの関係.....	39
1.3 カナ文字への対応.....	42

★ 2章 全システム・コール徹底解説 45

2.1 システム・コールとは.....	47
2.2 システム・コール徹底実習.....	51
ファンクション：0,1,2の実習.....	51
ファンクション：3,4,5の実習.....	54
ファンクション：6の実習.....	58
ファンクション：7,8の実習.....	60
ファンクション：9の実習.....	64
ファンクション：10の実習.....	66
ファンクション：11の実習.....	70
ファンクション：12の実習.....	72
ファンクション：13の実習.....	75
ファンクション：14,17,26の実習.....	77
ファンクション：15,20の実習.....	81
ファンクション：16,19,21,22の実習.....	85
ファンクション：18の実習.....	91
ファンクション：23の実習.....	94
ファンクション：24,25の実習.....	97
ファンクション：27の実習.....	101

ファンクション：28の実習	105
ファンクション：29の実習	108
ファンクション：30の実習	111
ファンクション：31の実習	114
ファンクション：32の実習	118
ファンクション：33の実習	121
ファンクション：34の実習	126
ファンクション：35の実習	139
ファンクション：36の実習	142
ファンクション：37の実習	143
ファンクション：40の実習	146
2.3 ランダム・アクセスによる検索プログラムの作成	149
2.3.1 プログラムの仕様	149
2.3.2 電子早見帳プログラムのアセンブリ・ソース・リスト	154
2.3.3 電子早見帳プログラムの実行	158
2.3.4 ディスク内容におけるデータの記録状態	162

★ 3章 マクロ・アセンブラおよびリンク・ローダによるソフト開発 163

3.1 MACとZSIDの使用例とマクロ・ライブラリの利用法	165
3.1.1 MACの機能	165
3.1.2 MACの使い方実例	165
3.1.3 シンボル・テーブルについて	178
3.1.4 SID, ZSIDの使用例	179
3.2 RMACの使用例	181
3.2.1 RMACの機能	181
3.2.2 RMACの使い方実例	181
3.3 MACRO-80によるモジュール別ソフト開発法とLINK-80	185
3.3.1 モジュール別ソフト開発法とは	185
3.3.2 MACRO-80の機能	186
3.3.3 MACRO-80によるモジュール別ソフト開発の実例	187
3.4 高級言語コンパイラとマシン語とのリンク	199
3.4.1 BASCOMによるメインプログラムの作成	200
3.4.2 アセンブラによる演算ルーチンの作成	202
3.4.3 メインプログラムと演算ルーチンとのリンク	203

★ 4章 他の8bit CPUおよび16bit CPUのソフト開発 205

4.1 ACT69の使用例	207
---------------	-----

4.2	XLT86の使用例	212
4.2.1	動作の概略	212
4.2.2	XLT86の実行例	213

★ 5章 各種高級言語による同一主題ソフト開発例 219

5.1	COBOL	222
5.1.1	COBOLについて	222
5.1.2	MICRO FOCUS社 CIS COBOLについて	223
5.1.3	CIS COBOLによる「SAMPLE」プログラムの作成	224
5.2	FORTRAN	229
5.2.1	FORTRANについて	229
5.2.2	Microsoft社 FORTRAN-80について	229
5.2.3	FORTRAN-80による「SAMPLE」プログラムの作成	230
5.3	BASIC	234
5.3.1	BASICについて	234
5.3.2	Digital Research社 CB-80について	234
5.3.3	CB-80による「SAMPLE」プログラムの作成	235
5.4	PASCAL	240
5.4.1	PASCALについて	240
5.4.2	Digital Research社 Pascal/MT+について	241
5.4.3	Pascal/MT+による「SAMPLE」プログラムの作成	241
5.5	PL/I	246
5.5.1	PL/Iについて	246
5.5.2	Digital Research社 PL/I-80について	247
5.5.3	PL/I-80による「SAMPLE」プログラムの作成	247
5.6	PL/M	251
5.6.1	PL/Mについて	251
5.6.2	Systems Consultants社 PLMXについて	252
5.6.3	PLMXによる「SAMPLE」プログラムの作成	252
5.7	C	259
5.7.1	Cについて	259
5.7.2	BD Software社 C Compilerについて	259
5.7.3	BDS Cによる「SAMPLE」プログラムの作成	260
5.8	FORTH	263
5.8.1	FORTHについて	263
5.8.2	Rgy FORTHについて	264
5.8.3	Rgy FORTHによる「SAMPLE」プログラムの作成	265

5.9	LISP	270
5.9.1	LISPについて	270
5.9.2	The Soft Warehouse社 muLISPについて	271
5.9.3	muLISPによる「SAMPLE」プログラムの作成	271
5.10	ALGOL	274
5.10.1	ALGOLについて	274
5.10.2	Mark Moranville ALGOL-Mについて	274
5.10.3	ALGOL-Mによる「SAMPLE」プログラムの作成	275
5.11	APL	278
5.11.1	APLについて	278
5.11.2	SOFTRONICS社 APL\80について	279
5.11.3	APL\80による「SAMPLE」プログラムの作成	279

★ 6章 CP/Mのアプリケーションいろいろ 285

6.1	簡易言語(プログラムレス言語)の使用例	287
6.1.1	SuperCalcの概念	287
6.1.2	SuperCalcの使用例	288
6.2	スクリーン・エディタの使用例	294
6.2.1	Micro Pro社のWord Master	294
6.2.2	Word Masterの実行例	295
6.3	スクリーン・オリエンテッドなソフトウェアとターミナルの適合について	300
6.3.1	エスケープ・シーケンスとは	300
6.3.2	自分のターミナルに適合させるためのスクリーン出力部の変更	301
6.3.3	パーソナル・コンピュータの場合	303
6.4	CP/MマシンとPROM書込器との接続	303
6.4.1	RS-232Cインターフェイスの接続	304
6.4.2	CP/Mで開発したプログラムのPROM書込器への転送	305
6.4.3	未知のPROM内データを読み出して、CP/Mで解析する例	306
6.5	CP/Mマシン間の音響カプラによる通信	308
6.5.1	専用プログラムを使用せず、PIPコマンドで伝送する例	309

あとがき	311
付録A NEOのCP/Mについて	312
付録B CP/M上で走るBASIC言語のステートメント・関数比較一覧表	314
付録C 本書で使用した各種ソフトウェアについて	319
付録D CP/M version2.2のバグについて	321
索引	323

システム・コール一覧 ●

ファンクション : 0	システム・リセット	51
ファンクション : 1	コンソールからの入力	51
ファンクション : 2	コンソールへの出力	52
ファンクション : 3	リーダーからの入力	54
ファンクション : 4	パンチへの出力	55
ファンクション : 5	リストへの出力	55
ファンクション : 6	ダイレクト・コンソール入出力	58
ファンクション : 7	IOバイトの取り出し	60
ファンクション : 8	IOバイトのセット	60
ファンクション : 9	文字列のプリントアウト	64
ファンクション : 10	コンソール・バッファへの読み込み	66
ファンクション : 11	コンソール入力ステータスのチェック	70
ファンクション : 12	オペレーティング・システムのバージョンNo.の取り出し	72
ファンクション : 13	ディスク・システムのリセット	75
ファンクション : 14	ディスク・ドライブの選択	77
ファンクション : 15	ファイルのオープン	81
ファンクション : 16	ファイルのクローズ	85
ファンクション : 17	最初のファイル・ディレクトリのサーチ	77
ファンクション : 18	次のファイル・ディレクトリのサーチ	91
ファンクション : 19	ファイルのデリート	85
ファンクション : 20	シーケンシャル・リード	81
ファンクション : 21	シーケンシャル・ライト	86
ファンクション : 22	ファイルの作成	86
ファンクション : 23	ファイル名の変更	94
ファンクション : 24	ログイン・ベクトルの取り出し	97
ファンクション : 25	ログイン・ディスクNo.の取り出し	97
ファンクション : 26	DMAアドレスのセット	78
ファンクション : 27	アロケーション・アドレスの取り出し	101
ファンクション : 28	ディスク・ライト・プロテクトのセット	105
ファンクション : 29	リード／オンリー・ベクトルの取り出し	108
ファンクション : 30	ファイル・アトリビュートのセット	111
ファンクション : 31	ディスク・パラメータ・アドレスの取り出し	114
ファンクション : 32	ユーザー・コードのセット／取り出し	118
ファンクション : 33	ランダムな読み出し	121
ファンクション : 34	ランダムな書き込み	126
ファンクション : 35	ファイル・サイズの計算	139
ファンクション : 36	ランダム・レコードのセット	142
ファンクション : 37	ディスク・ドライブのリセット	143
ファンクション : 40	ゼロ書き込み(ゼロ・フィル)を伴うランダムな書き込み	146

1章 CP/Mの内部構造と機能の詳細



本章では、CP/Mの内部構造を、CP/M上で実行する各種ソフトウェアを開発するユーザーや、CP/MのBIOSを変更して、独自のCP/Mに改造しようとするユーザーを対象に、具体的かつ詳細に、実践に役立つように解説します。本章は、次章の「全システム・コール徹底解説」とは密接な関係にありますので、それぞれ関係する項目を相互に参照して読み進んで下さい。

CP/Mの構成は、入門および実習CP/Mでも解説されているように、

TPA	(Transient Program Area)
CCP	(Console Command Processor)
BDOS	(Basic Disk Operating System)
BIOS	(Basic Input Output System)

と、アドレス0000H~00FFHのスクラッチ・パッド・エリアから成り立っています。そのメモリ上の配置を図示しておきます。「実習CP/M」の1章・2章も随時参照して下さい。

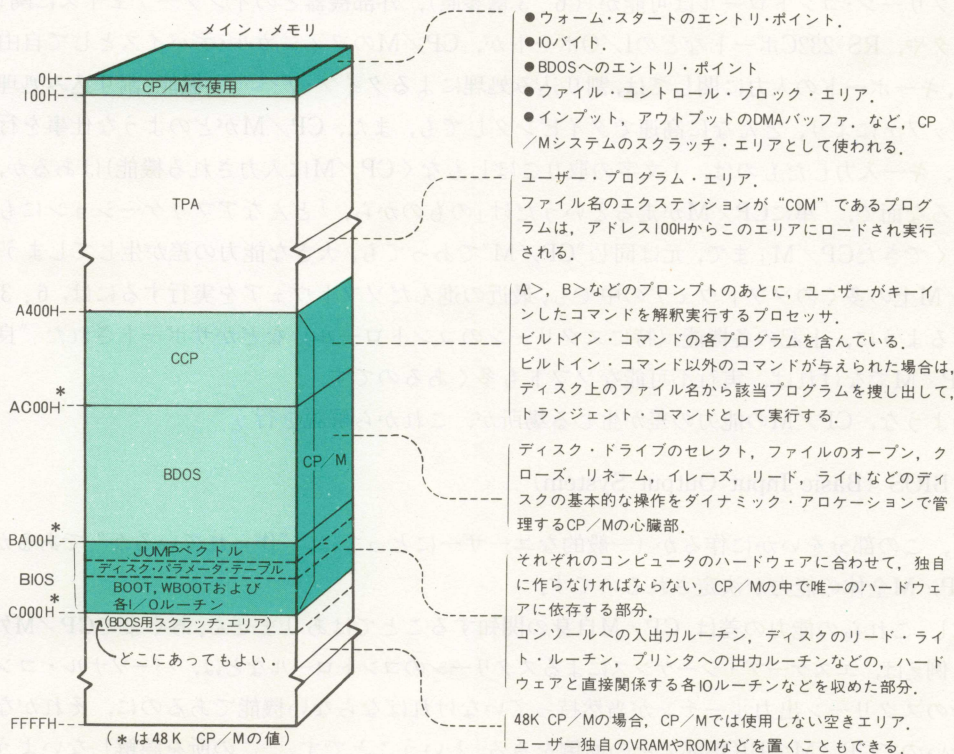


Figure-1.1.0 CP/Mシステムの構成

1.1 BIOS(基本入出力システム)

CP/Mは、どの機種でも能力は同じという訳ではありません。もちろんCPUのクロックが異なれば、処理スピードが違ってくるのは当然ですが、同一機種であっても、BIOSの作り次第で、例えばディスクの読み書きに要する時間などは、4～5倍の差が出てしまうのです。

ディスクのアクセス・スピードに関して言えば、8インチ片面単密度の“標準ディスク”では、ディスクの読み書きの際のスキュー・ファクタ（セクタの飛び越し数、後述）などが決められており、よほどへたな設計をしない限り、どれもほぼ同じスピードになります。しかし、倍密度や、4倍密度のフロッピー・ディスク、それにハード・ディスクになると、BIOSの設計の良し悪しで、4～5倍、あるいはそれ以上の差が出てしまいます。

良くできたCP/Mと、そうでないCP/Mの差は、ディスクのアクセス・スピードだけではありません。スクリーン表示に関しては、エスケープ・シーケンスによるカーソルのアドレッシングや、その他のスクリーン・コントロールは可能か（6.3章参照）、外部機器とのインターフェイスに関しては、プリンタや、RS-232CポートなどのI/Oポートが、CP/Mのフィジカル・デバイスとして自由に活用可能か、キーボードの入力に関しては、割り込み処理によるタイプ・アヘッド機能（割り込み処理とキー入力バッファにより、どんなに高速でタイピングしても、また、CP/Mがどのような仕事を行ってようと、キー入力したものは、1文字の取りこぼしもなくCP/Mに入力される機能）はあるか、など、いろいろな面で、「単にCP/Mが走るというだけ」のものから、「どんなアプリケーションにも対応できる良くできたCP/M」まで、元は同じ“CP/M”であっても、大きな能力の差が生じてしまうのです。

CP/M上の多くのソフトウェアの中でも、最近の進んだソフトウェアを実行するには、6.3章でも解説するように、上記の各機能（特にスクリーンのコントロール）などがサポートされた“良くできた”CP/Mでなければ、実行不可能なソフトも多くあるのです。

このような、CP/Mの能力の差が生じる場所が、これから解説を行う

BIOS (Basic Input Output System)

であり、この部分をいかに作るか（一般的なユーザーにとっては、“作られているか”であるが）により、CP/M全体の能力が決定されるのです。

ただし、これらの能力の差は、CP/M自身の関知することではありません。あくまでCP/M外の問題です。例えば、エスケープ・シーケンスによるスクリーンのコントロールなどは、パーソナル・コンピュータ自身のスクリーン出力ルーチンが当然持っていなければならない機能であるのに、それが無い。仕方がないのでCP/MのBIOS部でその面倒をみる、ということです。ここの所を誤解しないようにして下さい。

では、このBIOS部の構成、機能、作り方などの基本を解説して行きましょう。

1.1.1 BIOSの構成と機能

それぞれのコンピュータのハードウェアに合わせて作られたBIOSを、^{シーバイオス}“CBIOS”(Customized BIOS)と呼びます。ここでは、デジタルリサーチ社のマニュアルの中に、サンプルとして提示されているCBIOSのソース・ファイルの骨子を基に、解説を行います。

次に、CBIOSの構成をブロック図で示します。

BDOSとインターフェイス

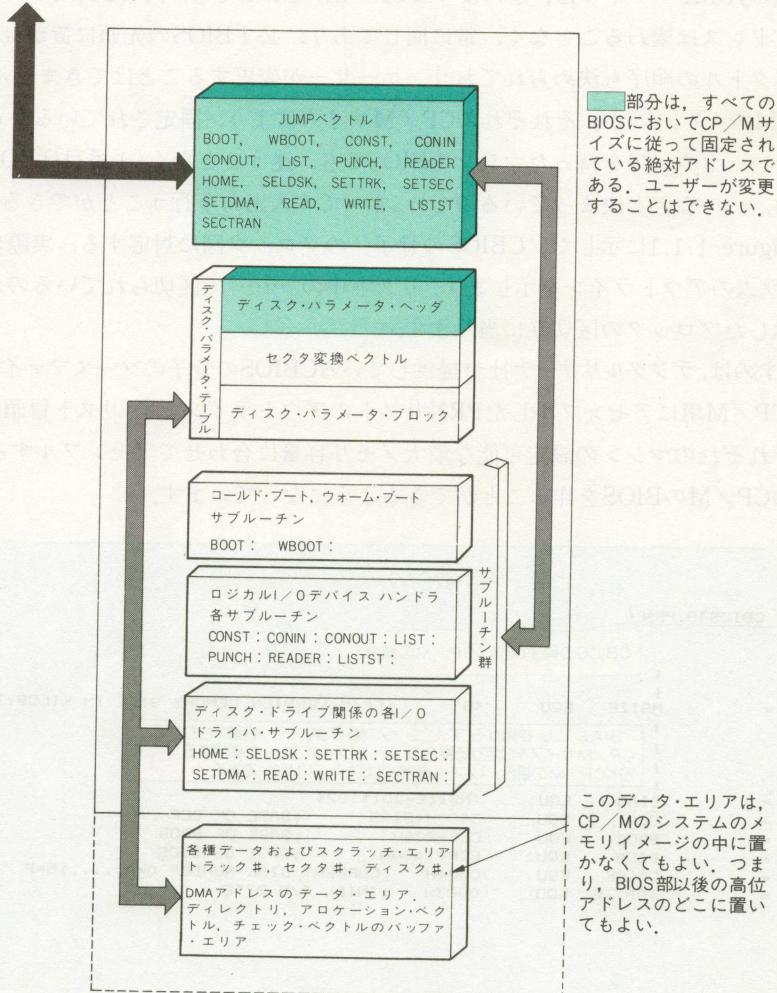


Figure-1.1.1 CBIOSの骨子

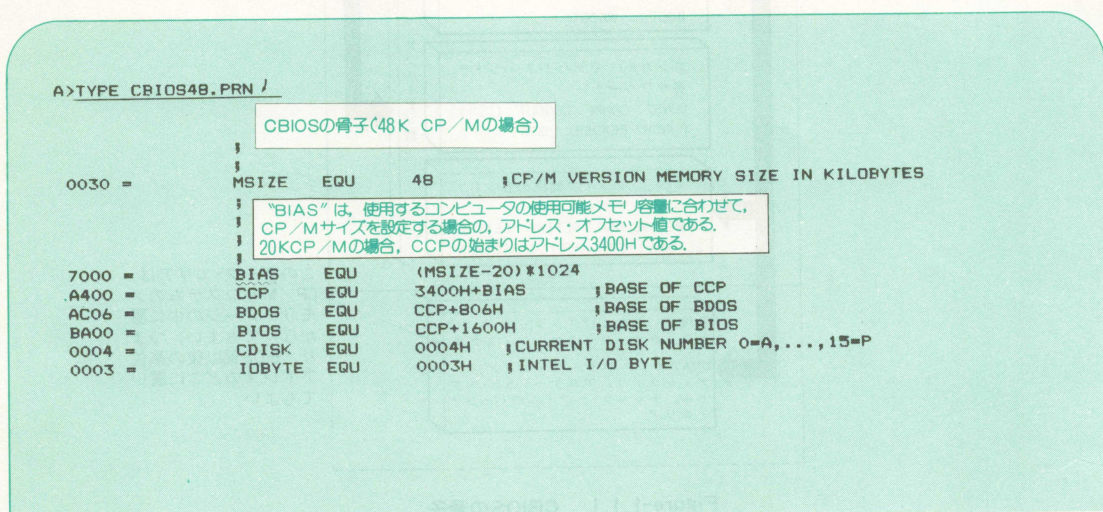
CP/Mのディスクを含めたすべての入出力は、BDOSにより、このBIOSを通して行われます。CP/Mが、何らかの入出力を行おうとする時、BDOSは、BIOSの先頭に集められているそれぞれのサブルーチンへ導くための“JUMPベクトル”の1つをコールし、そのベクトルに導かれて目的の機能が実行されます。また、入出力がディスク関係であれば、BDOSは、JUMPベクトルの次のブロックに位置する“ディスク・パラメータ・ヘッダ”を、JUMPベクトルの1つ“SELDISK”を介して参照し、ディスク管理に必要な各種データが格納されているアドレスを求め、そのディスクのデータを得ることによってすべてのディスク操作を行います。

合計17個のJUMPベクトルは、どのようなCP/MのBIOSでも、同じCP/Mサイズであれば、その位置するアドレスは変わることなく、常に同じであり、必ずBIOSの先頭に置かれています。同時に、それらのベクトルの順序も決められており、ユーザーが変更することはできません。

この“JUMPベクトル”が、それぞれのCP/Mサイズにより、固定されているからこそ、CP/Mは、どのようなハードウェアを持ったマシンのBIOSでも、迷うことなく(1番目はBOOT、4番目はコンソール入力、……などと決まっているので)、すべての入出力を行うことができます。

次に、Figure-1.1.1に示した“CBIOSの骨子”のブロック図に対応する、実際のアセンブリ・ソース・プログラムのアウトラインを示します。リスト中の“……”で区切られているのが、それぞれFigure-1.1.1に示したブロックの区切りに当たります。

ここに示すのは、デジタルリサーチ社が提供しているCBIOSの骨子のソースファイル、“CBIOS.ASM”を、48K CP/M用にアセンブルしたPRNリストのアウトラインです。リスト冒頭の、ラベル“MSIZE”の値を、それぞれのマシンの設定可能な最大メモリ容量に合わせてアセンブルすることにより、任意のサイズのCP/MのBIOSを作ることができるようになっています。



BA00 ORG BIOS ; ORIGIN OF THIS PROGRAM
002C EQU (\$-CCP)/128 ; WARM START SECTOR COUNT

すべての48 KCP/Mに
固定の絶対アドレス

それぞれのサブルーチンへのジャンプ・ベクトル
ジャンプ先のアドレスは、それぞれのCBIOSにより異なる。

BA00	C39CBA	JMP	BOOT	; COLD START
BA03	C3A6BA	WBOOT: JMP	WBOOT	; WARM START
BA06	C311BB	JMP	CONST	; CONSOLE STATUS
BA09	C324BB	JMP	CONIN	; CONSOLE CHARACTER IN
BA0C	C337BB	JMP	CONOUT	; CONSOLE CHARACTER OUT
BA0F	C349BB	JMP	LIST	; LIST CHARACTER OUT
BA12	C34DBB	JMP	PUNCH	; PUNCH CHARACTER OUT
BA15	C34FBB	JMP	READER	; READER CHARACTER OUT
BA18	C354BB	JMP	HOME	; MOVE HEAD TO HOME POSITION
BA1B	C35ABB	JMP	SELDSK	; SELECT DISK
BA1E	C37DBB	JMP	SETTRK	; SET TRACK NUMBER
BA21	C392BB	JMP	SETSEC	; SET SECTOR NUMBER
BA24	C3ADBB	JMP	SETDMA	; SET DMA ADDRESS
BA27	C3C3BB	JMP	READ	; READ DISK
BA2A	C3D6BB	JMP	WRITE	; WRITE DISK
BA2D	C34BBB	JMP	LISTST	; RETURN LIST STATUS
BA30	C3A7BB	JMP	SECTTRAN	; SECTOR TRANSLATE

CBIS の作り方に
より、各サブルー
チンのアドレスは
変化する。

8インチ片面密度の標準ディスクを使った、4ドライブ・システム用の
ディスク・パラメータ・テーブル

BA33	F3BA0000	DPBASE: DW	TRANS,0000H	DISK PARAMETER HEADER FOR DISK 00
BA37	00000000	DW	0000H,0000H	
BA3B	F0BC8DBA	DW	DIRBF,DPBLK	
BA3F	ECBD70BD	DW	CHK00,ALLO0	
BA43	73BA0000	DW	TRANS,0000H	DISK PARAMETER HEADER FOR DISK 01
BA47	00000000	DW	0000H,0000H	
BA4B	F0BC8DBA	DW	DIRBF,DPBLK	
BA4F	FCBD8FBD	DW	CHK01,ALLO1	
BA53	73BA0000	DW	TRANS,0000H	DISK PARAMETER HEADER FOR DISK 02
BA57	00000000	DW	0000H,0000H	
BA5B	F0BC8DBA	DW	DIRBF,DPBLK	
BA5F	OCBEAEBD	DW	CHK02,ALLO2	
BA63	73BA0000	DW	TRANS,0000H	DISK PARAMETER HEADER FOR DISK 03
BA67	00000000	DW	0000H,0000H	
BA6B	F0BC8DBA	DW	DIRBF,DPBLK	
BA6F	1CBECDBD	DW	CHK03,ALLO3	

これ以後のアドレスは、各
CBIOSにより一定ではない

BA73	01070D13	TRANS: DB	1,7,13,19	; SECTORS 1,2,3,4
BA77	19050B11	DB	25,5,11,17	; SECTORS 5,6,7,8
BA7B	1703090F	DB	23,3,9,15	; SECTORS 9,10,11,12
BA7F	1502080E	DB	21,2,8,14	; SECTORS 13,14,15,16
BA83	141A060C	DB	20,26,6,12	; SECTORS 17,18,19,20
BA87	1218040A	DB	18,24,4,10	; SECTORS 21,22,23,24
BA8B	1016	DB	16,22	; SECTORS 25,26
BA8D	1A00	DW	26	; DISK PARAMETER BLOCK, COMMON TO ALL DISKS
BA8F	03	DB	3	; SECTORS PER TRACK
BA90	07	DB	7	; BLOCK SHIFT FACTOR
BA91	00	DB	0	; BLOCK MASK
BA92	F200	DW	242	; NULL MASK
BA94	3F00	DW	63	; DISK SIZE-1
BA96	C0	DB	192	; DIRECTORY MAX
BA97	00	DB	0	; ALLOC 0
BA98	1000	DW	16	; ALLOC 1
BA9A	0200	DW	2	; CHECK SIZE
				; TRACK OFFSET

END OF FIXED TABLES

ディスク・パラメータ・ヘッド

これらのテーブルは、DISKDEFマクロ・ライブラリにより簡単に作成できる。

ディスク00〜03に共通
セクタ変換ベクトル
ディスク00〜03に共通
ディスクパラメータ・ブロック

<div> コード・ブート、ウォーム・ブートに関する一連のサブルーチン。 (注: “~”記号はユーザーによるルーチンを表す) </div>	
BA9C	BOOT: ;SIMPLEST CASE IS TO JUST PERFORM PARAMETER INITIALIZATION ~10バイト, ログイン・ディスクNo. などのイニシャライズ JMP GOCPM ; INITIALIZE AND GO TO CP/M
BAA6	WBOOT: ;SIMPLEST CASE IS TO READ THE DISK UNTIL ALL SECTORS LOADED ~コード・ブート・ローダとBIOS部を除く, システム・トラップ全体を読み出して, メモリにロードする。 ; END OF LOAD OPERATION, SET PARAMETERS AND GO TO CP/M GOCPM: ~ウォーム・ブートのための0Hのジャンプ・ベクトル, システム・コールのための5Hの BDOSへのジャンプ・ベクトルのセット, それにDMAアドレスのセットなど JMP CCP ;GO TO CP/M FOR FURTHER PROCESSING
<div> 4つのロジカル・デバイスの各ハンドラのサブルーチン。 それぞれのコンピュータのハードウェアに合わせて作成する。 この例では, 10バイトによるフィジカル・デバイスのサポートは行っていない。 </div>	
BB11	CONST: ;CONSOLE STATUS, RETURN OFFH IF CHARACTER READY, 00H IF NOT ~“CON:” のステータス・チェック・ルーチン。 RET
BB24	CONIN: ;CONSOLE CHARACTER INTO REGISTER A ~“CON:” の1文字入力ルーチン。 RET
BB37	CONOUT: ;CONSOLE CHARACTER OUTPUT FROM REGISTER C ~“CON:” の1文字出力ルーチン。 RET
BB49	LIST: ;LIST CHARACTER FROM REGISTER C ~“LST:” の1文字出力ルーチン。 RET
BB4B	LISTST: ;RETURN LIST STATUS (0 IF NOT READY, 1 IF READY) ~“LST:” のステータス・チェック・ルーチン。 RET
BB4D	PUNCH: ;PUNCH CHARACTER FROM REGISTER C ~“PUN:” の1文字出力ルーチン。 RET
BB4F	READER: ;READ CHARACTER INTO REGISTER A FROM READER DEVICE ~“RDR:” の1文字入力ルーチン。 RET
<div> ディスク・ドライブの入出力関係ドライブ・ルーチン </div>	
BB54	HOME: ;MOVE TO THE TRACK 00 POSITION OF CURRENT DRIVE TRANSLATE THIS CALL INTO A SETTRK CALL WITH PARAMETER 00 ~HOMEへのシーク・ルーチン。 RET ON FIRST READ/WRITE
BB5A	SELDSK: ;SELECT DISK GIVEN BY REGISTER C ~ドライブ選択ルーチン。 RET
BB7D	SETTRK: ;SET TRACK GIVEN BY REGISTER C ~トラックNo. セット・ルーチン。 RET
BB92	SETSEC: ;SET SECTOR GIVEN BY REGISTER C ~セクタNo. セット・ルーチン。 RET

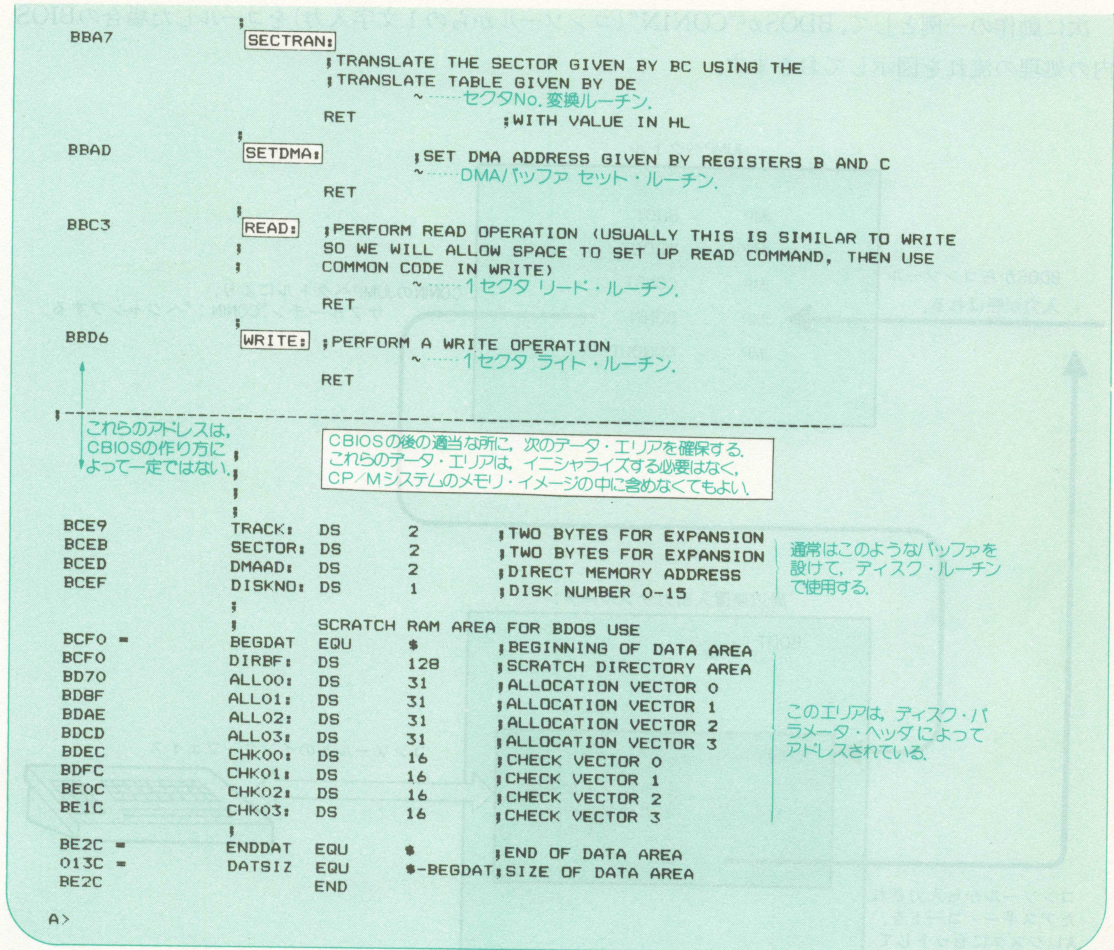


Figure-1.1.2 48K CP/MのCBIOSのソース・リストの骨子。

17個のJUMPベクトルによって導かれるそれぞれのサブルーチンを、リスト中では次のような形式で示しておきます。

ラベル名: ; このサブルーチンの機能
~(ユーザーによるルーチン)
RET

これらのサブルーチンは、CBIOSを作ろうとするユーザーが、それぞれのマシンに合わせて作らなければなりません。

次に動作の一例として、BDOSが"CONIN"(コンソールからの1文字入力)をコールした場合のBIOS内の処理の流れを図示しておきます。

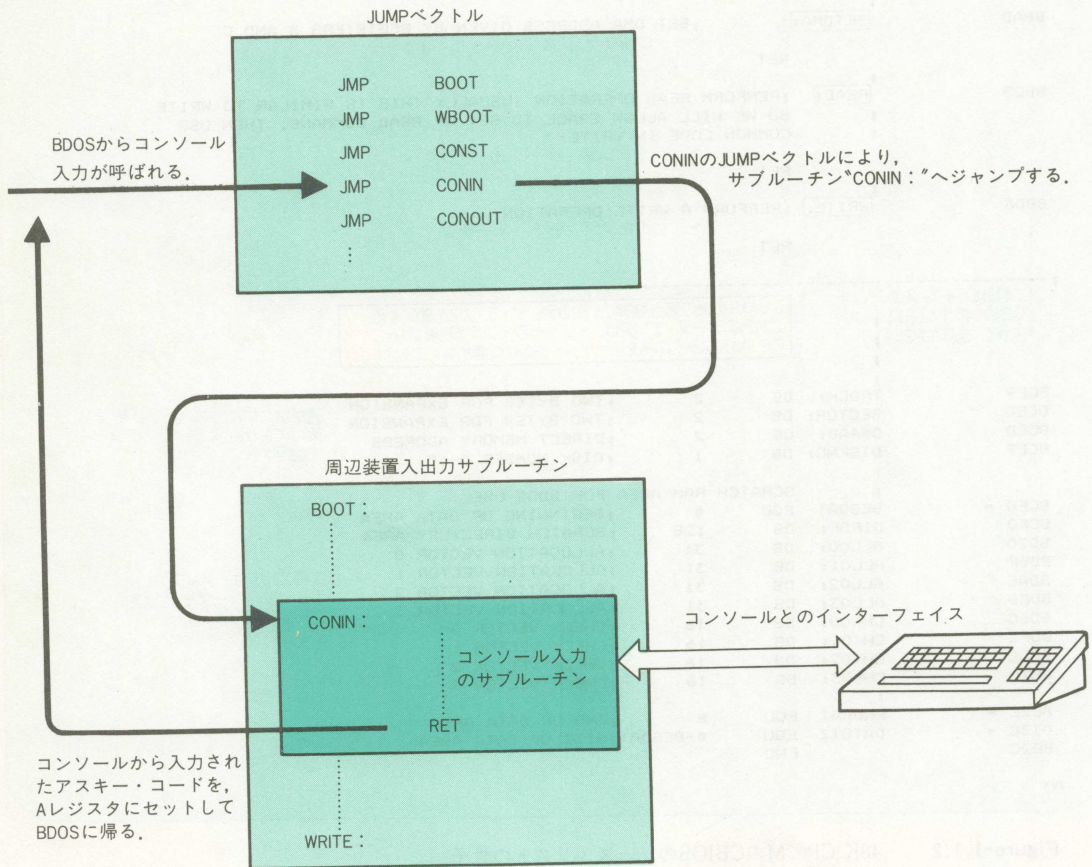


Figure-1.1.3 CBIOS内の処理の流れ

次に、各サブルーチンのそれぞれの機能の内容を解説します。

JUMPベクトルにより導かれる各サブルーチンの機能

各サブルーチンの入出力条件を、JUMPベクトルの並び順に示します。ここに示されているそれぞれの内容は、各ルーチンでの最低限必要な処理であり、他に必要があれば、ユーザー独自のものを拡張します。

BOOT: (コールド・ブート)

コールド・スタート・ローダから引き継がれ、システムの基本的なイニシャライズを行うルーチンである。次の“WBOOT”とも関連するので、共通のルーチン“GOCPM”を設け(リスト参照)、ここでは、IOバ이트のセットとドライブA:を選択するために、アドレス03H(ログイン・ディスクNo.の格納バ이트)を0にセットして、ラベル“GOCPM”へジャンプすればよい。その他ここで必要なものがあれば付け加え、あとの処理は“WBOOT”のルーチンと共通の“GOCPM”で行えばよい。

WBOOT: (ウォーム・ブート)

キーボードからのCtrl-Cの入力時や、アドレス0000Hへジャンプした時などのウォーム・スタート(リブート)の際に呼ばれるルーチンであり、次に示す各イニシャライズを行う。

- ディスクのシステム・トラックから、BIOSを除いたすべてを読み出し、メモリ上に再ロードする。
- アドレス00H~02Hに、ウォーム・スタートのジャンプ・ベクトルをセットする。
- アドレス05H~07Hに、システム・コールのためのBDOSへのジャンプ・ベクトルをセットする。
- ログイン・ディスクNo. (アドレス03Hの値)をCレジスタにセットして、CCPの先頭にジャンプする。

CONST: (コンソール・ステータス)

現在の“CON:”に割り当てられているデバイスのステータスをチェックする。READYであればFFH, BUSYであれば00HをAレジスタにセットしてリタンする。

CONIN: (コンソール・インプット)

現在の“CON:”に割り当てられているデバイスから1文字を入力し、そのアスキー・コードをAレジスタにセットしてリタンする。デジタルリサーチ社のマニュアルには、最上位ビットを0にするよう指示があるが、その必要はない(カナや、グラフィック・キャラクタに対応するには)。このルーチンは、入力がない場合は内部で入力があるまでループさせること。

CONOUT: (コンソール・アウトプット)

現在の“CON:”に割り当てられているデバイスに、Cレジスタにセットされているアスキー・コードを出力する。

LIST: (リスト・アウトプット)

現在の“LST:”に割り当てられているデバイスに、Cレジスタにセットされているアスキー・コードを出力する。

PUNCH: (パンチ・アウトプット)

現在の“PUN:”に割り当てられているデバイスに、Cレジスタにセットされているアスキー・コードを出力する。

READER: (リーダ・インプット)

現在の“RDR:”に割り当てられているデバイスから1文字を入力し、Aレジスタにセットしてリタンする。入力がない場合は、入力されるまで内部でループする。これも、“CONIN”の場合と同様に、最上位ビットを0にする必要はない。

HOME: (ホームへのシーク)

ログイン・ディスクのヘッドを、ホーム位置(トラック00)にシークする。後述の“SETTRK”ルーチンを利用してもよい。

SELDSK: (セレクト・ディスク・ドライブ)

CレジスタにセットされているドライブNo. (A:=0, B:=1, ……P:=15)のドライブを選択するためのルーチン。

CレジスタにセットされているドライブNo.に従って、そのドライブのディスク・パラメータ・ヘッダのベース・アドレス(リストでは、それぞれ、BA33H, BA43H, BA53H, BA63Hに当たる)をH, Lレジスタにセットしてリタンする。もし、存在しないドライブが選択された場合は、H, Lレジスタに0000Hをセットしてリタンする。実際の動作は、リード/ライトの直前に行えばよい(注)。

SETTRK: (セット・トラック)

B, Cレジスタ(フロッピー・ディスクであれば、256トラック以下であるので、Cレジスタのみ見ればよい)にセットされている値のトラックNo.に、ヘッドをシークする。実際の動作は、リード/ライトの直前に行えばよい(注)。

SETSEC: (セット・セクタ)

B, Cレジスタにセットされている値のセクタNo.を、ディスク・コントローラに送り込む。実際の動作は、リード/ライトの直前に行えばよい(注)。

SETDMA: (セットDMAアドレス)

B, Cレジスタにセットされているアドレス(デフォルトは0080H, その他の値は, ユーザーのシステム・コールによる設定で任意)に, ディスクのリード/ライトを行う際の128バイトの入出力バッファを設定する.

要するに, ここで設定されたバッファを介して, リード/ライト時のデータの入出力が行われる.

READ: (ディスク・リード)

"SELDISK", "SETTRK", "SETSEC"で, あらかじめ指定されている1セクタのデータを読み出し, DMAバッファに格納する.

セクタ・リードが正常に行われた場合は00, もし回復できないエラーを起こした場合は01のエラー・コードをAレジスタにセットしてリタンする. ただし, エラーが起こった場合, 普通は10回程度のリトライを行う.

エラー・コードにより, BDOSはエラー・メッセージを出力する.

WRITE: (ディスク・ライト)

DMAバッファの1セクタのデータを, ディスクに書き込む. その他は上記"READ"の場合と全く同じ.

LISTST: (リスト・ステータス)

現在の"LST:"に割り当てられているデバイスのステータスをチェックする. READYならFFH, BUSYなら00HをAレジスタにセットしてリタンする.

バック・グラウンド・プリントアウト・プログラムの"DESPool"などでこのルーチンが利用される. 通常は00Hとなるようにしておくこと.

SECTTRAN: (セクタ・トランスレータ)

ディスクのリード/ライトを少しでも高速に行う目的の, スキュー(セクタの飛び越し読み書き, 後述)を行うための, ロジカル・セクタ⇒フィジカル・セクタの変換ルーチンである.

ロジカル・セクタNo.がB, Cレジスタに, セクタ変換テーブルの先頭アドレスがD, Eレジスタにセットされ, 当ルーチンがコールされる. 変換されたフィジカル・セクタNo.は, H, Lレジスタにセットしてリタンする.

注) これらのパラメータを, Figure-1.1.2のアドレスBCE9H~BCEFhのバッファにセットしておき, リード/ライトの直前に一括して実行すればよい.

ここで示したCBIOSのブロック図や、ソース・リスト、各ルーチンの機能などは、標準的なCP/Mを動作させるための必要最少限のBIOSであり、IOバイトによるロジカル・デバイスに対する各フィジカル・デバイスの選択やそれらのドライバ・ルーチンなどは含まれていません。

IOバイトを利用する場合は、4つのロジカル・デバイスの各ルーチン（ステータス・チェック・ルーチンも含む）のそれぞれの冒頭に、アドレス03HのIOバイトの値をみて、その値が指示するデバイスを選択するための“デバイス・セレクト・ルーチン”を設けなくてはなりません。その上で、それらのフィジカル・デバイス自身のルーチンを設ける訳です。

その他、“良くできた使えるCP/M”にするためには、オプション的な各種機能を、それぞれのルーチンに盛り込む必要もあるでしょう。一方、フィジカル・デバイスや、ロジカル・デバイスの中で、もし使う必要がない周辺装置があるならば、それらのサブルーチンは書く必要はありません。その場合は、もし誤ってコールされた場合の暴走を防ぐために、“RET” 命令を書いておき、何もせずにリターンするようにしておくのが良いでしょう。

ユーザー・プログラムから各エントリ・ポイントの直接CALL

前述の17のJUMPベクトルは、“BOOT”と“WBOOT”を除き、それらに続くルーチンは、最後が“RET” 命令で終わるサブルーチンの形式をとっています。よって、2章で解説する“システム・コール”を利用しなくても、JUMPベクトルに導かれる17の機能は、このエントリ・ポイントに対する各CP/Mサイズに固有の絶対アドレスを、直接にCALLすることにより、それぞれ実行することができます。“BOOT”、“WBOOT”のルーチンは、サブルーチンではありませんが、スタック・ポインタもイニシャライズしますので、“CALL” 命令でも“JMP” 命令でもどちらでも実行可能です。

この BIOSエントリ・ポイントの直接CALLによるメリットは、

- ① コンソールなどの周辺装置の入出力に関するものでは、システム・コールに伴うBDOS内でのオーバーヘッド（目的の機能が実行されるまでに介在する各種ルーチン）が省かれるので、その分だけ処理が高速になる。
- ② BDOSに一切関係せず各処理が行える。
- ③ システム・コールの機能にはない、任意のセクタのリード／ライトが可能である。

などが挙げられます。

CALLの具体的な方法は、CALLに際しパラメータが必要なものは、それぞれのレジスタにパス・パラメータをセットして、それぞれのJUMPベクトルの絶対アドレスをCALLすればよい訳です。

例えばFigure-1.1.2の48K CP/Mでは、

Cレジスタ←出力キャラクタ

CALL BA0CH

を実行することにより、コンソールに任意の1文字を出力することができます。

しかし、CP/Mのサイズは20K~64Kまで様々ですので、実際のアプリケーションには、どのCP/Mサイズに対しても実行可能となるように、アドレス0000H~0002Hにある"WBOOT"のエントリ・ポイントに示されているアドレスを基に、その他16のJUMPベクトル自身のアドレスを算出するというテクニックがよく使われています。

次に、BIOS JUMPベクトル・エントリ・ポイントの一覧表を示しておきます。

BOOT=4A00(20K), 7A00(32K), BA00(48K), FA00(64K)

アドレス	ベクトル名	機 能	バス・パラメータ	リターン・パラメータ
××00H	BOOT	コールド・フート	—	C←0
××03H	WBOOT	ウォーム・ブート	—	C←ドライブNo. A←FFH/00 (レディー/ビジー)
××06H	CONST	コンソール・ステータス・チェック	—	A←入力キャラクタ
××09H	CONIN	コンソール入力	—	—
××0CH	CONOUT	コンソール出力	C←出力キャラクタ	—
××0FH	LIST	リスト出力		—
××12H	PUNCH	パンチ出力		—
××15H	READER	リーダ入力	—	A←入力キャラクタ
××18H	HOME	ヘッドのホーム・シーク	—	—
××1BH	SELDISK	ディスク・ドライブの選択	C←ドライブNo.	HL←DPH
××1EH	SETTRK	トラックNo.のセット	C←トラックNo.	—
××21H	SETSEC	セクタNo.のセット	C←セクタNo.	—
××24H	SETDMA	DMAアドレスのセット	BC←DMAアドレス	—
××27H	READ	指定されたセクタのリード	—	A←00/01 (OK/エラー)
××2AH	WRITE	// ライト	—	—
××2DH	LISTST	リスト・ステータスのチェック	—	A←FFH/00 (レディー/ビジー)
××30H	SECTAN	セクタ・トランスレータ	BC←ロジカル・セクタNo. DE←トランスレータ ・テーブル・アド レス	HL←フィジカル・ セクタNo.

BIOS JUMPベクトル・エントリ・ポイント

1.1.2 ディスク・パラメータ・テーブル

CBIOSのブロック図や、ソース・リストに示されているように、"JUMPベクトル"に続くエリアは、"ディスク・パラメータ・テーブル"と呼ばれる、ディスクの入出力を行う際に必要な、各ディスク・ドライブに関する情報を保管するためのエリアが置かれています。

ディスク・パラメータ・テーブルは、次に示す3つの部分から構成されています (Figure-1.1.1およびFigure-1.1.2も参照)。

1. ディスク・パラメータ・ヘッダ
2. セクタ・トランスレート・テーブル
3. ディスク・パラメータ・ブロック

これらのそれぞれについて説明して行きましょう。2章の「システム・コール」のファンクション27, 31, 35などとも密接な関係がありますので、相互に参照して下さい。

ディスク・パラメータ・ヘッダ

DPHと呼ばれ、CBIOSのリストに示されているように、4ドライブ・システムであれば4個、nドライブ・システムであればn個というように、各ドライブにそれぞれ独立したDPHを設けます。

DPHの構成を、Figure-1.1.2のCBIOSのリストを例に解説しましょう。

リストでは、1個のドライブの各パラメータを、1ラインに4バイトずつ、4ラインに分けて書いてありますが、ここでは横一列に並べて示します。

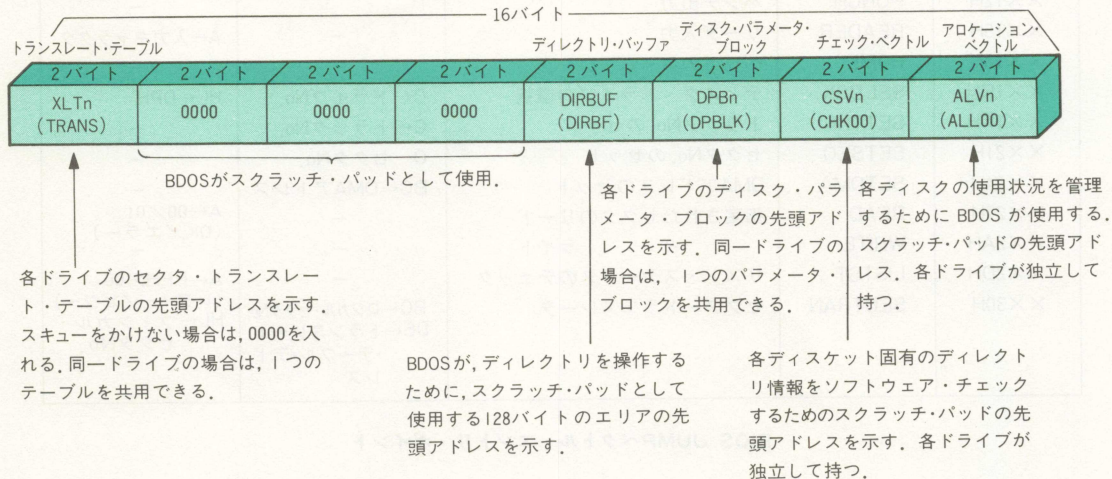
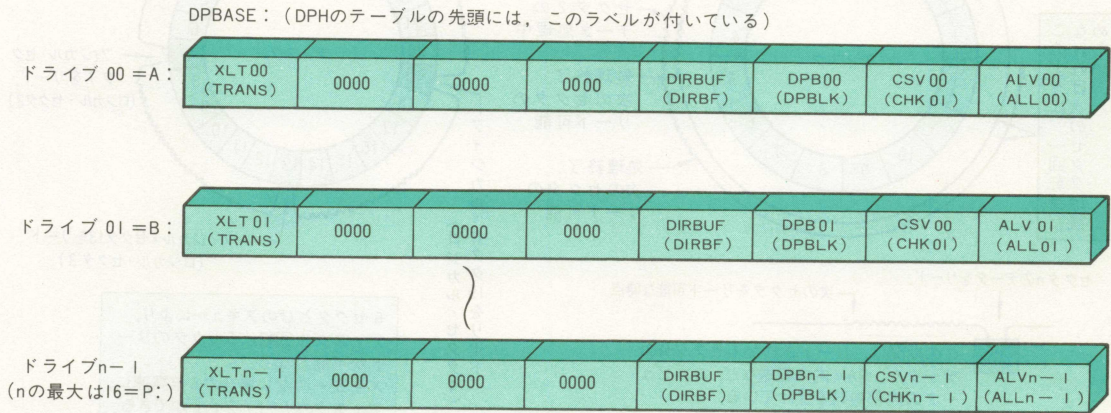


Figure-1.1.4 1つのドライブのディスク・パラメータ・ヘッダの構成

上図で、()内は、Figure-1.1.2のCBIOSのリストの“DISK00”の場合のシンボル名です。この場合の実アドレスが、リスト左側に示されていますので、実際のトランスレート・テーブル(TRANS:)や、リスト最後の“SCRATCH RAM AREA”とを対比して下さい。

図中の小文字の n は、ディスク No. を示しますが、同一機種のドライブを使用するのであれば、“XLT” と “DPB” は 1 つのものを全部のドライブで共用することができます (CBIOS のリストでは、共用している)。ただし、“CSV”、“ALV” については完全に独立していなければなりません。

ディスク・ドライブを n 個使用するシステムでは、Figure-1.1.4 のディスク・パラメータ・ヘッダを、次の図に示すように、それぞれのドライブに対応して n 個設けます。



図中で、() のついてないシンボルの呼び方は、後述の “DISKDEF” マクロ・ライブラリで用いられているものであり、() 中の呼び方は、Figure-1.1.2 の CBIOS のリストでのものである。

CBIOS のリストでは、4 つの同一ドライブを使用するため、“TRANS” と “DPBLK” は、それぞれ 1 つ設けるだけで、すべてのドライブで共用している。

Figure-1.1.5 n 個のディスク・ドライブを持ったシステムのディスク・パラメータ・ヘッダ

注) このディスク・パラメータ・ヘッダは、後述の “DISKDEF” マクロ・ライブラリにより、自動的に作成できます。

セクタ・トランスレート・ベクトル

同心円状の各トラックのセクタを、少しでも速く読み書きするための手法として、“スキュー”があります。スキューは、セクタの飛び越し読み書きのことであり、普通にアクセスする場合は、同一トラック上のセクタを、ディスク 1 回転につき 1 セクタしかリード／ライトできないところ、セクタ番号にこのスキューを持たせることにより、数セクタのリード／ライトを可能にするものです。

この “スキュー” の概念を次の図で示します。

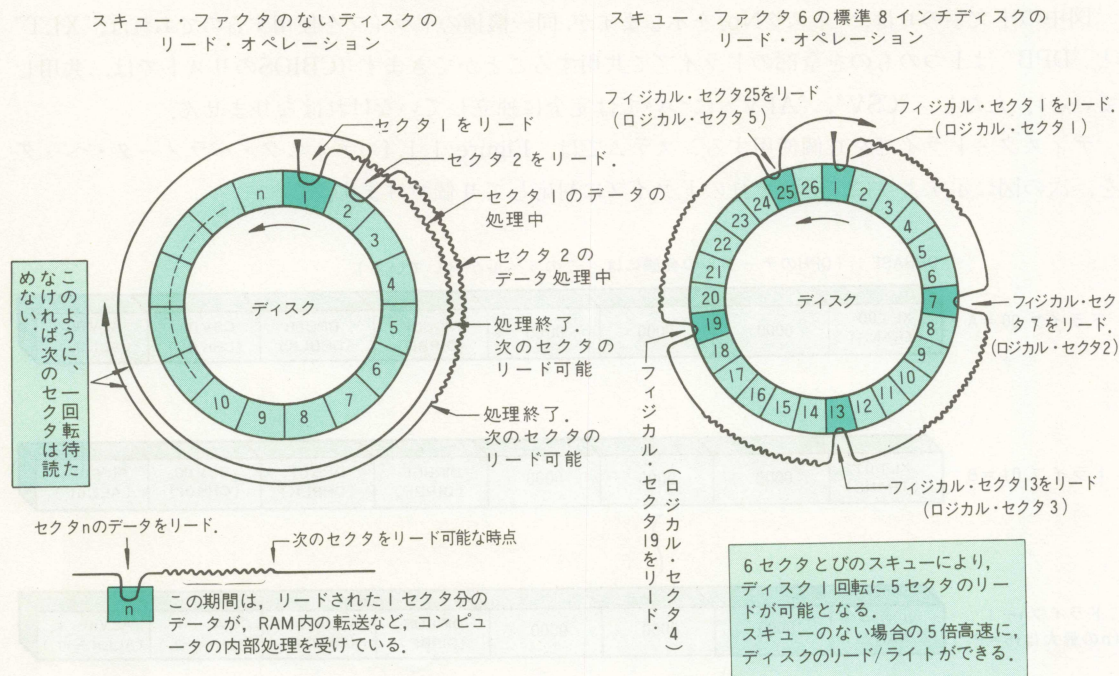


Figure-1.1.6 スキューの概念

上図は、スキューのない場合と、ある場合、ディスク上の1つのトラックをリードする際の模式図です。

まず、スキューのない場合、セクタを1, 2, 3, ……のように順番に読み出すことを考えてみましょう。実際のディスクのアクセスは、このようなセクタを順番に連続してリード/ライトする場合（シーケンシャル・ファイル）が多いのです。現実には、ディスクが回転するのですが、図では便宜上、ヘッドがディスクの回りを回転しながら読み出すように説明されています。

図に示されているように、ディスクが回転し、リード/ライト・ヘッドの下にセクタ1が来た時に、セクタ1のデータが読み出されました。そのデータを、所定のメモリに格納し、次のセクタであるセクタ2を読み出す準備が整った時には、すでにセクタ2は、ヘッドの下を通り過ぎてしまっています。ディスクが1回転するのを待たなければ、セクタ2を読み出すことはできません。つまり、スキューがなければ、セクタを連続して読む場合は、ディスク1回転につき1セクタしかリードできないということを示しています。

次に、スキュー・ファクタ6を持つ、8インチ片面単密度の標準フロッピー・ディスクの場合を考えてみましょう。セクタ変換テーブルの実際は、Figure-1.1.2のCBIOSのリストにも示されていますが、CP/Mが、標準ディスクのロジカル・セクタ（論理セクタ）を1, 2, 3, ……と順番に読むということは、

実際には、フィジカル・セクタ（物理セクタ）を、1, 7, 13, 19, ……と読み出すことになるのです。

さて、この場合のリード・オペレーションの様子がFigure-1.1.6に図示されていますが、このようにディスク1回転に、ロジカル・セクタ1, 2, 3, 4, 5の計5セクタが読み出されています。先程の、スキューがない場合の5倍の速さでディスクをアクセスできる訳です。図では“リード”に対して述べていますが、“ライト”の場合も全く同様にこれらのことが言えます。

以上で、“スキュー”の概念は理解されたと思います。そして標準ディスクのように、スキュー・ファクタを持った場合に必要となるものが、本題である「セクタ・トランスレート・ベクトル」である訳です。

8インチ標準ディスクのセクタ変換テーブルは、Figure-1.1.2のCBIOSのリストに、その実際のものが示されていますが、ロジカル・セクタとフィジカル・セクタは次のように対応しています。

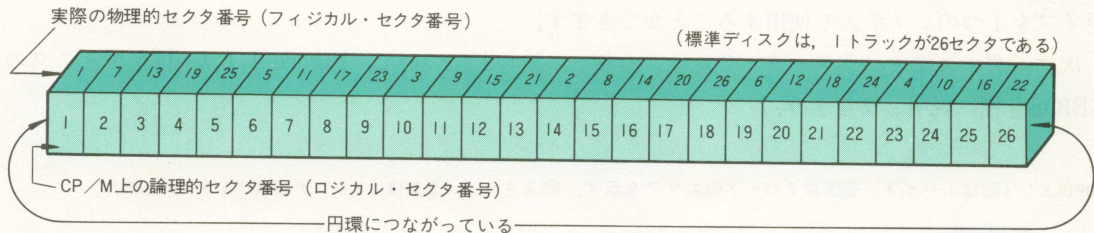


Figure-1.1.7 8インチ標準ディスクのロジカル・セクタ⇒フィジカル・セクタの対応

ディスクのリード／ライトを行う時、その管理者であるBDOSは、ロジカル・セクタ番号をCBIOSに与え、そのフィジカル・セクタ番号を“SECTTRAN” サブルーチンで計算させます。そこで得られたフィジカル・セクタ番号を、“SETSEC” サブルーチンで指定して、“READ” や “WRITE” を実行させる訳です。

注) このセクタ・トランスレート・ベクトルは、スキュー・ファクタを指定することにより、後述の“DISKDEF” マクロ・ライブラリにより、自動的に作成されます。

スキューのないディスクはすべてアクセスが遅いという訳ではありません。ハードウェアでスキューを持たせているものや、ブロック・リード／ライト（ディスク1回転に1トラックの全部のデータをリード／ライトするもの）を行って高速化を計ることもできます。

ディスク・パラメータ・ブロック

ディスク・パラメータ・ブロック (DPB) は、Figure-1.1.2のCBIOSのリストでは、ラベル“DPBLK”で始まる一連のテーブルであり、ディスク・パラメータ・ヘッダの“DPBn” (Figure-1.1.2のリストでは“DPBLK”) によって、そのブロックの先頭がアドレスされています。

DPBは、BDOSがディスク管理をするのに必要な、各ドライブのハードウェア的な情報のすべてを持っており、ミニディスクから8インチ・ディスク、それにハード・ディスクまで、すべてのドライブは、このDPBによってその諸元が定義され、BDOSに管理されます。

Figure-1.1.2のCBIOSのリストでは、4つのドライブが同一の機種であるため、1つのDPBを共有しています。しかし、CP/Mは、1つのシステムで、機種の異なるディスク・ドライブを自由に組み合わせ使用することができます。

例えば、ドライブA：がミニの両面ドライブ、ドライブB：がミニの片面ドライブとなるシステム(このようなシステムは、両面⇄片面のファイル変換が、PIPコマンドで自由にできる)とか、ドライブA：、B：が8インチの両面で(ただし、8インチの片面単密ディスクをセットした場合は、自動判別して、そのドライブは、8インチ片面として動作するようにしておくといふ)、ドライブE：、F：がミニの両面というように、それぞれのドライブに専用のDPBを設けることにより、このような異なったドライブを1つのシステムで使用することができます。

次に、ディスク・パラメータ・ブロックの各フィールドについて、Figure-1.1.2の標準ディスクのCBIOSを例に説明しましょう。

DPBLK： (■は1バイト、■は2バイトのエリアを示す。記入されている数値は8インチ標準ディスクのもの)

SPT	26	Sectors Per Track, 1トラックあたりの全ロジカル・セクタ数 (128バイトを1セクタとする)。																																																
BSH	03	Block Shift factor, データアロケーション・ブロックシフトのファクタで、データ・ブロック・アロケーションサイズによって決められる。数値“3”の意味は、 $1024=128 \times 2^3$ の“3乗”のこと。																																																
BLM	07	Block Mask, ブロック・マスクのことで、データ・ブロックのアロケーション・サイズによって決められる。数値“7”の意味は、0～7の8セクタ=1024バイトで1ブロックを示す。																																																
EXM	00	Extent Mask, データ・ブロックのアロケーション・サイズと、ディスク・ブロック数によって決められるエクステント・マスク。																																																
DSM	242	Disk Size Max, ディスクのデータ・ブロックの最大数-1。数値“242”の意味は、 $\{(128 \text{ バイト} \times 26 \text{ セクタ}) \times (77 \text{ 本} - 2 \text{ 本}) / 1024 \text{ バイト}\} - 1 = 242$ (余りは切り捨て)。																																																
DRM	63	Directory entries Max, ディスクに収容可能なディレクトリ・エントリの数-1。 次のAL0とAL1はこのディレクトリ・ブロックによって決められる。																																																
AL0	192	<table><tr><th colspan="8">AL 0</th><th colspan="8">AL 1</th></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td colspan="8">00</td><td colspan="8">15</td></tr></table> ←AL0=192=C0H, AL1=0をセットしたビット・パターン。	AL 0								AL 1								1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00								15							
AL 0								AL 1																																										
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																			
00								15																																										
AL1	0	各ビットは、ディレクトリ・エントリに使用するデータ・ブロックを示す。この例では、2つのビットが1なので、2つのデータ・ブロック=2Kバイトがディレクトリ・エリアとして使用されることを示している。																																																
CKS	16	Check vector Size, ディレクトリをチェックするベクトルのサイズ。 $CKS=(DRM+1)/4$ 。ただし、ハード・ディスクのようなメディアを交換しないものは0となる。																																																
OFF	2	track Offset, システム・トラックをスキップするためのオフセット値。あるいは、大容量のハード・ディスクを、複数のドライブに分割する場合にも利用できる。数値“2”は、システム・トラックが2本であることを示す。																																																

このDPBの各パラメータ間の関係を次に示します。

B L S (ブロック)	B S H	B L M
1,042	3	7
2,048	4	15
4,096	5	31
8,192	6	63
16,384	7	127

Figure-1.1.8-1

B L S	D S M (255以下)	D S M (256以上)
1,024	0	—
2,048	1	0
4,096	3	1
8,192	7	3
16,384	15	7

Figure-1.1.8-2

B L S	Iドライブに収容可能な ディレクトリ・エントリ総数
1,024	(AL0, AL1のビット"1"の数) ×32
2,048	×64
4,096	×128
8,192	×256
16,384	×512

Figure-1.1.8-3

注) これらのディスク・パラメータ・ブロックは、後述の“DISKDEF”マクロ・ライブラリにより、自動的に作成することができます。

1.1.3 DISKDEFマクロ・ライブラリの使い方

前項1.1.2で述べた3つの部分から成るディスク・パラメータ・テーブルと、CBIOS最後部のスクラッチRAMエリアは、標準CP/Mのディスクケットに含まれているファイル、“DISKDEF.LIB”(ディスク・ディファイン・マクロ・ライブラリ)を、デジタルリサーチ社のマクロ・アセンブラの“MAC”でアセンブルすることにより、自動的に作成することができます。

“MAC”については、3.1章を参照して下さい。

DISKDEFマクロは、一般的に、CBIOSの中の次に示す位置に置かれます。

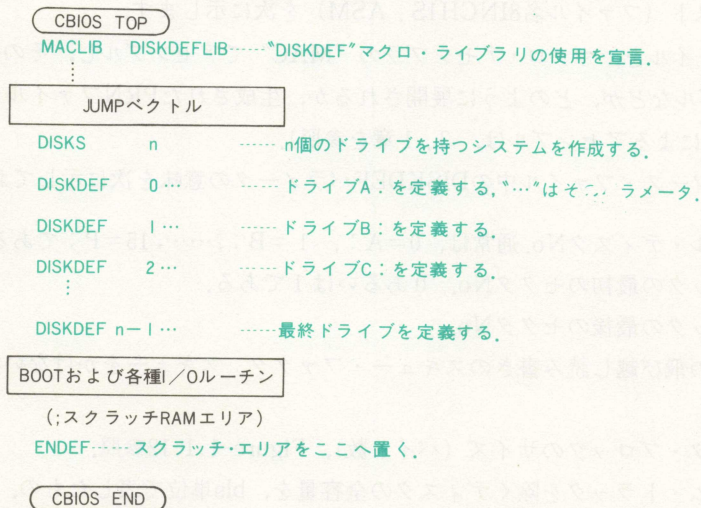


Figure-1.1.9 DISKDEFマクロの位置

次に、DISKDEFマクロ・ライブラリを使って、8インチ標準ディスクの場合と、5インチのミニ画面ディスクの場合について、実際にパラメータを設定してアセンブルを行い、ディスク・パラメータ・テーブルなどを自動的に作成してみましょう。

8インチ片面単密度の標準ディスクの場合

このディスクの物理的パラメータは、

128バイト／1セクタ
26セクタ／1トラック
77トラック／1ディスク

であり、標準ディスクのシステム・ディスクは、

2本のシステム・トラック
64のディレクトリ・エントリ数

となっていますので、これらから、DISKDEFの各パラメータを決定します。

アセンブルするためのソース・ファイルは、本来ならば、CBIOS全体である訳ですが、ここでは、DISKDEFマクロ・ライブラリだけに注目してもらうため、DISKDEFマクロのみのソース・ファイルを作りました。そのアドレスは、Figure-1.1.2のCBIOSのリストと一致するように、ORGで指定してあります。

そのソース・リスト（ファイル名8INCH1S.ASM）を次に示します。

このソース・ファイルを、マクロ・アセンブラの“MAC”でアセンブルし、その結果、ディスク・パラメータ・テーブルなどが、どのように展開されるか、生成されたPRNファイルを見てみることにしましょう（MACによるアセンブルは、3.1章を参照）。

その前に、上記ソース・ファイル中のDISKDEFパラメータの意味を次に示しておきます。

dn …ロジカル・ディスクNo. 通常は、0=A:、1=B:、……、15=P:である。

fsc …各トラックの最初のセクタNo. 0あるいは1である。

lsc …各トラックの最後のセクタNo.

skf …セクタの飛び越し読み書きのスキュー・ファクタ。スキューをかけない場合は、これを省略する。

bls …1データ・ブロックのサイズ（バイト数）、Figure-1.1.12参照。

dks …システム・トラックを除くディスクの全容量を、bls単位で表したもの。

dir …ディレクトリ・エントリの数。

cks …書き込み前にログインされてからディスクが交換されていないかをチェックするディレクトリ・エントリの数。フロッピー・ディスクのように、メディアが交換できるものはdirと同じ値。ハード・ディスクのように変換しないものは0をセットする(チェック不要)。

ofs …システム・トラックを確保するためのオフセット数。通常は、使用するシステム・トラックの本数をセットする。ハード・ディスクの場合は、ドライブ単位に分割するのにも利用される。

```
A>TYPE BINCH1S.ASM J
;***** 8 Inch Single Sided *****

MACLIB DISKDEF .....DISKDEFマクロ・ライブラリの使用を宣言。
ORG      OBA33H .....(Figure-1.1.2のリストと同じアドレスにするため。)

DISKS 4 .....4ドライブのシステムを作成。
      dn  isc  bis  dir  ofs
DISKDEF 0,1,26,6,1024,243,64,64,2]
      fsc  skf  dks  cks
DISKDEF 1,0
DISKDEF 2,0
DISKDEF 3,0
;
;
;
;-----
;
;      SCRATCH RAM AREA FOR BDOS USE
;
;      ORG      OBCFOH .....(Figure-1.1.2のリストと同じアドレスにするため)。
;
;      ENDEF .....スクラッチRAMエリアをここに設ける。
;
;      END
;
A>
```

ディスク1,2,3を、ディスク0と同じパラメータに設定する。この記述の一般形は“n,j”であり、n番目のディスクを、j番目と同じパラメータに設定する場合に使用する。共用可能なテーブルは共用されるので、メモリの節約になる。

Figure-1.1.10 標準ディスク用のディスク・パラメータ・テーブルを、“DISKDEF”マクロ・ライブラリで実験的に作成するためのソース・ファイル。

以上9個のパラメータを、Figure-1.1.10に示されているように標準ディスクの場合の値にセットして、DISKDEFのマクロ・コールを行います。

MACによるアセンブル後のPRNファイルを次に示します。展開された4つの部分を、Figure-1.1.2のCBIOSのリストと照合してみてください。

A>TYPE 8INCH1S.PRN)

;***** 8 INCH SINGLE SIDED *****

BA33 MACLIB DISKDEF
 ORG 0BA33H

DISKS 4

		EQU	\$	
BA33+=	DPBASE	EQU	\$; BASE OF DISK PARAMETER BLOCKS
BA33+82BA0000	DPE0:	DW	XLTO,0000H	; TRANSLATE TABLE
BA37+00000000		DW	0000H,0000H	; SCRATCH AREA
BA3B+F0BC73BA		DW	DIRBUF,DPB0	; DIR BUFF,PARM BLOCK
BA3F+8FBD70BD		DW	CSV0,ALV0	; CHECK, ALLOC VECTORS
BA43+82BA0000	DPE1:	DW	XLT1,0000H	; TRANSLATE TABLE
BA47+00000000		DW	0000H,0000H	; SCRATCH AREA
BA4B+F0BC73BA		DW	DIRBUF,DPB1	; DIR BUFF,PARM BLOCK
BA4F+BEED9FBD		DW	CSV1,ALV1	; CHECK, ALLOC VECTORS
BA53+82BA0000	DPE2:	DW	XLT2,0000H	; TRANSLATE TABLE
BA57+00000000		DW	0000H,0000H	; SCRATCH AREA
BA5B+F0BC73BA		DW	DIRBUF,DPB2	; DIR BUFF,PARM BLOCK
BA5F+EDBDCEBD		DW	CSV2,ALV2	; CHECK, ALLOC VECTORS
BA63+82BA0000	DPE3:	DW	XLT3,0000H	; TRANSLATE TABLE
BA67+00000000		DW	0000H,0000H	; SCRATCH AREA
BA6B+F0BC73BA		DW	DIRBUF,DPB3	; DIR BUFF,PARM BLOCK
BA6F+1CBEFDBD		DW	CSV3,ALV3	; CHECK, ALLOC VECTORS

ディスク・パラメータ・ヘッダ

DISKDEF 0,1,26,6,1024,243,64,64,2

		EQU	\$	
BA73+=	DPB0	EQU	\$; DISK PARM BLOCK
BA73+1A00		DW	26	; SEC PER TRACK
BA75+03		DB	3	; BLOCK SHIFT
BA76+07		DB	7	; BLOCK MASK
BA77+00		DB	0	; EXTNT MASK
BA7B+F200		DW	242	; DISK SIZE-1
BA7A+3F00		DW	63	; DIRECTORY MAX
BA7C+C0		DB	192	; ALLOC0
BA7D+00		DB	0	; ALLOC1
BA7E+1000		DW	16	; CHECK SIZE
BA80+0200		DW	2	; OFFSET

ディスク・パラメータ・ブロック

		EQU	\$	
BA82+=	XLTO	EQU	\$; TRANSLATE TABLE
BA82+01		DB	1	
BA83+07		DB	7	
BA84+0D		DB	13	
BA85+13		DB	19	
BA86+19		DB	25	
BA87+05		DB	5	
BA88+0B		DB	11	
BA89+11		DB	17	
BA8A+17		DB	23	
BA8B+03		DB	3	
BA8C+09		DB	9	
BA8D+0F		DB	15	
BA8E+15		DB	21	
BA8F+02		DB	2	
BA90+0B		DB	8	
BA91+0E		DB	14	
BA92+14		DB	20	
BA93+1A		DB	26	
BA94+06		DB	6	

セクタ変換ベクトル


```

BA95+0C      DB      12
BA96+12      DB      18
BA97+18      DB      24
BA98+04      DB       4
BA99+0A      DB      10
BA9A+10      DB      16
BA9B+16      DB      22

DISKDEF 1,0
BA73+=       DPB1    EQU    DPB0    ;EQUIVALENT PARAMETERS
001F+=       ALS1    EQU    ALSO    ;SAME ALLOCATION VECTOR SIZE
0010+=       CSS1    EQU    CSS0    ;SAME CHECKSUM VECTOR SIZE
BAB2+=       XLT1    EQU    XLT0    ;SAME TRANSLATE TABLE

DISKDEF 2,0
BA73+=       DPB2    EQU    DPB0    ;EQUIVALENT PARAMETERS
001F+=       ALS2    EQU    ALSO    ;SAME ALLOCATION VECTOR SIZE
0010+=       CSS2    EQU    CSS0    ;SAME CHECKSUM VECTOR SIZE
BAB2+=       XLT2    EQU    XLT0    ;SAME TRANSLATE TABLE

DISKDEF 3,0
BA73+=       DPB3    EQU    DPB0    ;EQUIVALENT PARAMETERS
001F+=       ALS3    EQU    ALSO    ;SAME ALLOCATION VECTOR SIZE
0010+=       CSS3    EQU    CSS0    ;SAME CHECKSUM VECTOR SIZE
BAB2+=       XLT3    EQU    XLT0    ;SAME TRANSLATE TABLE

;
;
;
;-----
;          SCRATCH RAM AREA FOR BDOS USE

BCF0          ORG      0BCF0H

ENDEF
BCF0+=        BEGDAT  EQU    $
BCF0+=        DIRBUF: DS      128    ;DIRECTORY ACCESS BUFFER
BD70+=        ALV0:   DS      31
BD8F+=        CSV0:   DS      16
BD9F+=        ALV1:   DS      31
BDBE+=        CSV1:   DS      16
BDCE+=        ALV2:   DS      31
BDED+=        CSV2:   DS      16
BDFD+=        ALV3:   DS      31
BE1C+=        CSV3:   DS      16
BE2C+=        ENDDAT  EQU    $
013C+=        DATSIZ  EQU    $-BEGDAT

BE2C          END

```

スクラッチRAMエリア

Figure-1.1.11 DISKDEFマクロ・ライブラリにより、自動的に作成された標準ディスクのディスク・パラメータ・テーブル。

このように、Figur-1.1.2のリストとは、ラベルの名称や配置が異なるものの、ディスク・パラメータ・テーブルの内容は全く一致していることが分かります。

5 インチの両面ディスクの場合

5 インチの両面ディスクについて、先程と同様にディスク・パラメータ・テーブルを作成してみましょう。

ミニの両面ディスク・ドライブの物理的な諸元は、次のようなものとします。

256バイト／1セクタ

32セクタ／1トラック（サイド0，サイド1の両面で）

40トラック／1ディスク

このディスクには，“スキュー”をかけないことにします。そして，DISKDEFパラメータのデータ・ブロックのサイズ“bls”は2Kバイトにして2048，bls単位で表したディスクの全容量“dks”は152，ディレクトリ・エントリの数“dir”と“cks”は128とします。1トラック当りのロジカル・セクタ数“lsc”は， $32 \times 256 / 128 = 64$ となります。オフセット数“ofs”は前回と同じ2とします。

では，前回のFigure-1.1.10のソース・リストと同じものに，上記のパラメータをセットしたものを次に示します。ファイル名は“MINI2W.ASM”です。

```

A>TYPE MINI2W.ASM
;***** Mini Double Sided *****
MACLIB DISKDEF
ORG OBA33H
DISKS 4
DISKDEF 0,1,64,,2048,152,128,128,2
DISKDEF 1,0
DISKDEF 2,0
DISKDEF 3,0
;
;
;
;-----
; SCRATCH RAM AREA FOR BDOS USE
;
ORG OBCF0H
ENDEF
END
A>

```

スキューをかけないのでスキップ
していることに注意。

Figure-1.1.12 ミニの両面ディスク用のディスク・パラメータ・テーブルを，“DISKDEF”マクロ・ライブラリで実験的に作成するためのソース・ファイル。

これをMACを使ってアセンブルし、生成されたPRNファイルを次に示します。展開の状態を、Figure -1.1.11のものと比較して下さい。

A>TYPE MINI2W.PRN)

***** MINI DOUBLE SIDED *****

BA33	MACLIB	DISKDEF	
	ORG	OBA33H	
DISKS 4			
BA33+=	DPBASE	EQU	\$; BASE OF DISK PARAMETER BLOCKS
BA33+00000000	DPE0:	DW	XLT0,0000H ; TRANSLATE TABLE
BA37+00000000		DW	0000H,0000H ; SCRATCH AREA
BA3B+F0BC73BA		DW	DIRBUF,DPB0 ; DIR BUFF,PARM BLOCK
BA3F+83BD70BD		DW	CSV0,ALV0 ; CHECK, ALLOC VECTORS
BA43+00000000	DPE1:	DW	XLT1,0000H ; TRANSLATE TABLE
BA47+00000000		DW	0000H,0000H ; SCRATCH AREA
BA4B+F0BC73BA		DW	DIRBUF,DPB1 ; DIR BUFF,PARM BLOCK
BA4F+B6BDA3BD		DW	CSV1,ALV1 ; CHECK, ALLOC VECTORS
BA53+00000000	DPE2:	DW	XLT2,0000H ; TRANSLATE TABLE
BA57+00000000		DW	0000H,0000H ; SCRATCH AREA
BA5B+F0BC73BA		DW	DIRBUF,DPB2 ; DIR BUFF,PARM BLOCK
BA5F+E9BDD6BD		DW	CSV2,ALV2 ; CHECK, ALLOC VECTORS
BA63+00000000	DPE3:	DW	XLT3,0000H ; TRANSLATE TABLE
BA67+00000000		DW	0000H,0000H ; SCRATCH AREA
BA6B+F0BC73BA		DW	DIRBUF,DPB3 ; DIR BUFF,PARM BLOCK
BA6F+1CBEO9BE		DW	CSV3,ALV3 ; CHECK, ALLOC VECTORS

DISKDEF 0,1,64,,2048,152,128,128,2

BA73+=	DPB0	EQU	\$; DISK PARM BLOCK
BA73+4000		DW	64 ; SEC PER TRACK
BA75+04		DB	4 ; BLOCK SHIFT
BA76+0F		DB	15 ; BLOCK MASK
BA77+01		DB	1 ; EXTNT MASK
BA7B+9700		DW	151 ; DISK SIZE-1
BA7A+7F00		DW	127 ; DIRECTORY MAX
BA7C+C0		DB	192 ; ALLOC0
BA7D+00		DB	0 ; ALLOC1
BA7E+2000		DW	32 ; CHECK SIZE
BA80+0200		DW	2 ; OFFSET

0000+=	XLT0	EQU	0 ; NO XLATE TABLE
DISKDEF 1,0			
BA73+=	DPB1	EQU	DPB0 ; EQUIVALENT PARAMETERS
0013+=	ALS1	EQU	ALSO ; SAME ALLOCATION VECTOR SIZE
0020+=	CSS1	EQU	CSS0 ; SAME CHECKSUM VECTOR SIZE
0000+=	XLT1	EQU	XLT0 ; SAME TRANSLATE TABLE
DISKDEF 2,0			
BA73+=	DPB2	EQU	DPB0 ; EQUIVALENT PARAMETERS
0013+=	ALS2	EQU	ALSO ; SAME ALLOCATION VECTOR SIZE
0020+=	CSS2	EQU	CSS0 ; SAME CHECKSUM VECTOR SIZE
0000+=	XLT2	EQU	XLT0 ; SAME TRANSLATE TABLE
DISKDEF 3,0			
BA73+=	DPB3	EQU	DPB0 ; EQUIVALENT PARAMETERS
0013+=	ALS3	EQU	ALSO ; SAME ALLOCATION VECTOR SIZE
0020+=	CSS3	EQU	CSS0 ; SAME CHECKSUM VECTOR SIZE
0000+=	XLT3	EQU	XLT0 ; SAME TRANSLATE TABLE

ディスク・パラメータ・ヘッダ

ディスク・パラメータ・ブロック

セクタ変換ベクトル
が作られていないこ
とに注意。

; .			
; .			
; .			
; -----			
; SCRATCH RAM AREA FOR BDOS USE			
BCF0	ORG	OBCF0H	
ENDEF			
BCF0+=	BEGDAT	EQU	\$
BCF0+	DIRBUF:	DS	128 ; DIRECTORY ACCESS BUFFER
BD70+	ALV0:	DS	19
BD83+	CSV0:	DS	32
BDA3+	ALV1:	DS	19
BDB6+	CSV1:	DS	32
BDD6+	ALV2:	DS	19
BDE9+	CSV2:	DS	32
BE09+	ALV3:	DS	19
BE1C+	CSV3:	DS	32
BE3C+=	ENDDAT	EQU	\$
O14C+=	DATSIZ	EQU	\$-BEGDAT
BE3C	END		
A>			

スラッシュRAMエリア

Figure-1.1.13 DISKDEFマクロ・ライブラリにより、自動的に作成された、ミニの両面ディスクのディスク・パラメータ・テーブル。

今回はスキューがないため、セクタ変換ベクトルが作られていません。また、先程の8インチ標準ディスクのものと、アセンブル結果の各数値が異なることを、よく比較して下さい。

データ・ブロックについて

ここで、“データ・ブロック”について、解説しておきましょう。次に示す図の(A)は、8インチの標準ディスクの場合を表し、(B)は、ミニの両面ディスクの場合の一例を表しています。

1データ・ブロックとは、Figure-1.1.10では、DISKDEFパラメータの“bls”に当たるものであり、本図(A)では128バイトが8個(8セクタ)連続した計1024バイト=1Kバイトのブロックに当たります。

“データ・ブロック”は、CP/Mが管理するディスク上のファイル・データの最少単位であり、これより小さい容量のファイルは作ることができません。例えば(A)の場合は、1ページ(256バイト)のSAVEを行っても、それによって作られたファイルは1Kバイト長となる訳です。

(B)の場合は、物理的な1セクタである256バイトが8個連続した計2048バイト=2Kバイトのブロックが、1データ・ブロックであり、ファイルの最少単位は2Kバイトということになります。

(A)の場合、システム・トラックを除く全ディスク容量は、243データ・ブロックで243Kバイトであり、(B)の場合は、152データ・ブロックで304Kバイトであることなども、本図から理解されると思います。

(A)

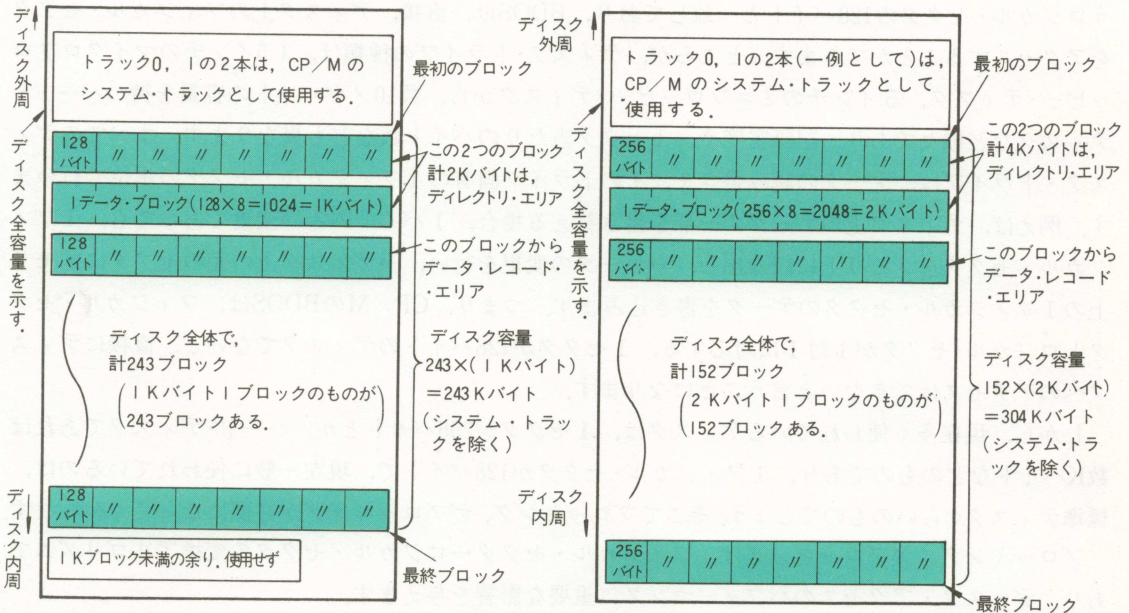
8 インチ片面単密度ディスクの場合のデータ・ブロック

(128バイト/1セクタ, 26セクタ/1トラック, 全77トラック)

(B)

5 インチ両面ミニディスクの場合のデータ・ブロック

(256バイト/1セクタ, 32セクタ/1トラック, 全40トラック)



$$\frac{(128 \text{ バイト} \times 26 \text{ セクタ}) \times (77 \text{ 本} - 2 \text{ 本})}{1024 \text{ バイト} (= 1 \text{ データ・ブロック})} = 243.75 \text{ ブロック}$$

(余りは切り捨てる)

$$\frac{(256 \text{ バイト} \times 32 \text{ セクタ}) \times (40 \text{ 本} - 2 \text{ 本})}{2048 \text{ バイト} (= 1 \text{ データ・ブロック})} = 152 \text{ ブロック}$$

Figure-1.1.14 データ・ブロック

データ・ブロックは、CP/Mが管理するディスク上のファイルの最少単位であり、各ファイルのディスク上のアロケーションは、各ファイルを構成しているデータ・ブロックすべてについて、ディスク上のどこに散在しているかを、常にディレクトリ・エリアに記録することにより管理されています。

ディスク上の物理的に隣り合うデータ・ブロックが、1つのファイルの連続であるとは限らず、別のファイルの一部であったりする訳ですが、1つのデータ・ブロックの中では、シーケンシャル・ファイルであれば連続しています。つまり1データ・ブロックが2Kバイトであれば、2Kバイト分のデータの内部はディスクのロジカル・セクタ上で連続している訳です。

この“データ・ブロック”は、CP/Mのディスク入出力のすべての基になるものであり、よく理解しておく必要があります。

1.1.4 セクタ・ブロッキング、デブロッキングの概念

8 インチ片面単密度の標準ディスクは、物理的な1セクタが128バイトであり、CP/MのBDOSが扱うロジカル・セクタの128バイトと一致しており、BDOSは、直接、ディスク上のフィジカル・セクタをアクセスすることができます。ところが、ディスク・ドライブの種類は、3.5インチのマイクロフロッピー・ディスク、5インチのミニフロッピー・ディスクから、数10メガバイトの容量を持つハード・ディスクなど様々であり、記録密度や、1セクタ当たりのバイト数なども異なります。すべてのディスク・ドライブは、データの読み書きを、そのドライブ固有の1フィジカル・セクタの単位で行います。例えば、ディスク上のデータの一部を書き替える場合、1バイトのみの変更であっても、1フィジカル・セクタをメモリ上に読み出し、→データの変更を行い、→ディスク上の元のセクタにメモリ上の1フィジカル・セクタのデータを書き込みます。つまり、CP/MのBDOSは、フィジカル・セクタとロジカル・セクタが1対1に対応する、1セクタが128バイトのディスクでないと、直接にディスクへのアクセスはできないということになります。

しかし、現在多く使われているディスクは、1セクタが256バイトとか、ハード・ディスクであれば数Kバイトなどのものであり、1フィジカル・セクタが128バイトで、現在一般に使われているのは、標準ディスクぐらいのものでしょう。そこでブロッキング、デブロッキングが必要になってくるのです。

ブロッキング、デブロッキングは、フィジカル・セクタ→ロジカル・セクタの変換アルゴリズムであり、ディスク・アクセスのパフォーマンスに重要な影響を与えます。

まず、ブロッキング、デブロッキングの原理を図示したものを次に示します。この例は、ディスクのフィジカル・セクタが512バイトであり、OSのロジカル・セクタは、CP/Mを想定しているので当然128バイトの場合について解説しています。

ブロッキングは、ディスクへデータを書き込む場合（OSの一般用語では“PUT”）の手法であり、デブロッキングはその逆の、ディスクからデータを読み出す場合（“GET”）の手法のことを言います。そしてこれらのルーチンは、BIOS内に設けられます。

まず、デブロッキングから解説しましょう。

デブロッキング

Figure-1.1.15の「デブロッキングの原理」から、もうすでに概念は把握されていることと思います。が、この図を基に説明しましょう。

例えば、“TYPE”コマンドなどで、ディスク上のファイルをシーケンシャルに読み出す場合を考えて下さい。

BDOSは、最初のセクタ（最初のロジカル・セクタの128バイト）をディスクから読み出したいのですが、ディスクからは128バイト単位では読み出すことはできません。そこでBIOSは、ロジカル・セ

クタ1が含まれる、ディスク上のフィジカル・セクタ1を、RAM上のホスト・バッファに読み出します。

ホスト・バッファは、1フィジカル・セクタのバイト数の容量を持ち、これは、1ロジカル・セクタのバイト数の整数倍に当たります。図の例では、1フィジカル・セクタは512バイトですので、ホスト・バッファは、 $128 \times 4 = 512$ バイトの容量を持っています。

フィジカル・セクタ1の512バイトのデータは、一度のディスク・アクセスで瞬時にしてホスト・バッファに格納されました。BDOSは、さっそくホスト・バッファ内のロジカル・セクタ1の128バイトのデータを処理し、次のロジカル・セクタ2のデータを読み込もうとします。ここで重要なのは、BDOSがロジカル・セクタ2を読み込もうとする時に、ディスク上のフィジカル・セクタ1を再度アクセスしないということです。求めるロジカル・セクタ2は、RAM上のホスト・バッファに存在しており、これを処理する方が、再度ディスクをアクセスするよりずっと速いからです。

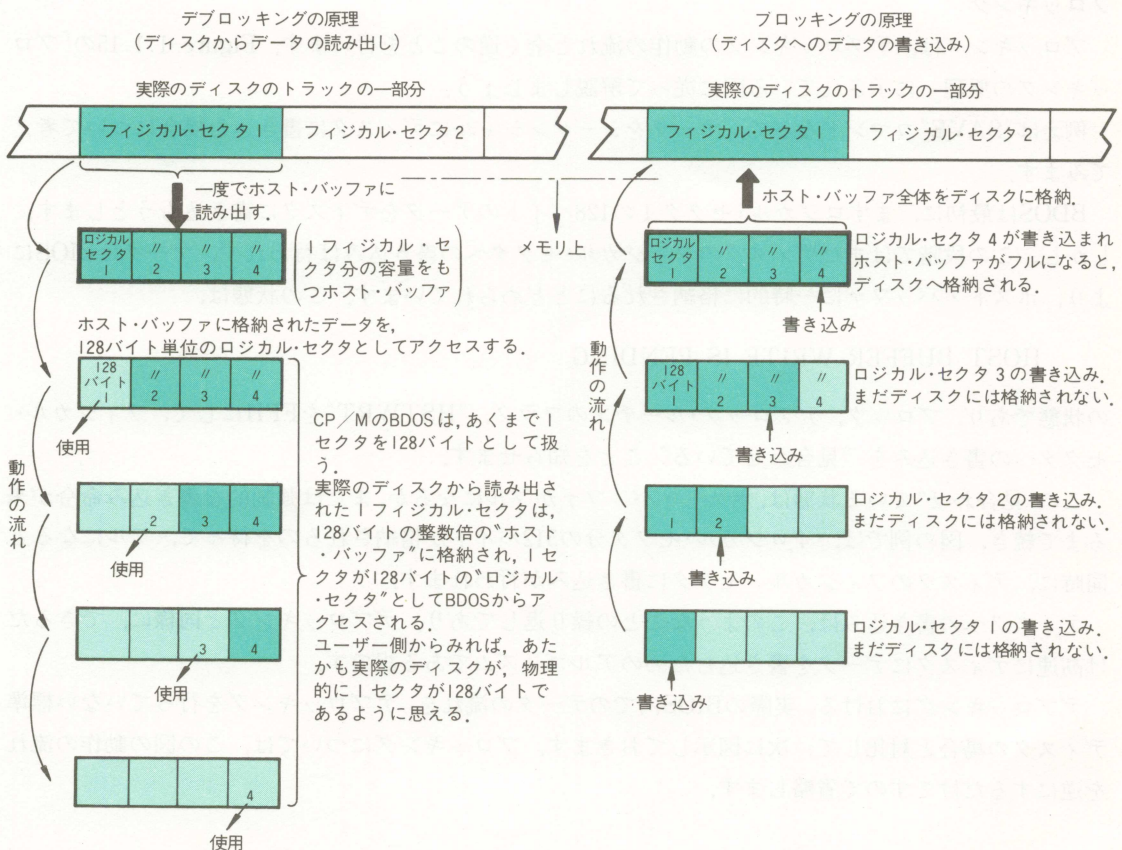


Figure-1.1.15 ブロッキング、デブロッキングの原理

このように、ロジカル・セクタ2～4は、ディスクを再度アクセスすることなく、ホスト・バッファを介してBDOSにより処理されます。

この図の例で、BDOSがロジカル・セクタ1を読もうとして、BIOSが自動的にフィジカル・セクタ1をホストバッファに読み出した時点を、

HOST BUFFER DATA IS ACTIVE

と言い、ブロック・デブロック・ルーチンのフラグ“HSTACT”をFFHにして、ロジカル・セクタ1～4に関しては、ディスクをアクセスする必要のないことを知らせます。

デブロッキングは、このような手順を各フィジカル・セクタで繰り返し、できるだけ高速にディスクからデータの読み出しを行うためのアルゴリズムである訳です。

ブロッキング

ブロッキングは、デブロッキングの動作の流れと全く逆のことを行います。Figure-1.1.15の「ブロッキングの原理」に示されている図に従って解説しましょう。

例えば“SAVE”コマンドなどで、データをシーケンシャルにディスクに書き込む場合について考えてみます。

BDOSは最初に、まずロジカル・セクタ1の128バイトのデータをディスクに書き込もうとします。しかし、この段階ではまだディスクのフィジカル・セクタへの書き込みは行われず、データはBIOSにより、ホスト・バッファに一時的に格納されるにとどめられています。この状態は、

HOST BUFFER WRITE IS PENDING

の状態であり、ブロック、デブロック・ルーチンのフラグ，“HSTWRT”をFFHにして、フィジカル・セクタへの書き込みを“見合わせている”ことを知らせます。

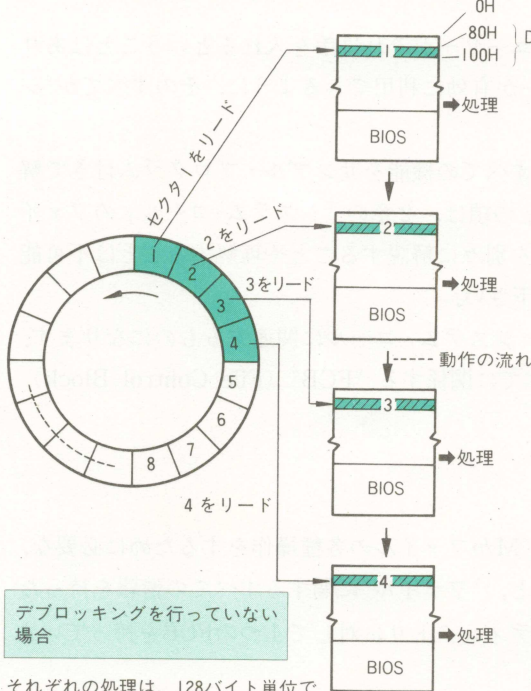
この“見合わせ”ている状態は、ホスト・バッファがフルになるか、または強制的な書き込み命令が来るまで続き、図の例では、4ロジカル・セクタ分の512バイトが格納されるのを待って、フルになると同時に、ディスクのフィジカル・セクタに書き込みが行われます。

ディスクへの書き込みは、このようなことの繰り返しであり、デブロッキングと同様に、できるだけ高速にディスクにデータを書き込むためのアルゴリズムである訳です。

デブロッキングにおける、実際のBIOS内でのデータの流れを、デブロッキングを行っていない標準ディスクの場合と対比して、次に図示しておきます。ブロッキングについては、この図の動作の流れを逆にするだけでするので省略します。

単密度ディスク（物理的な1セクタが128バイト）のシーケンシャルな読み出し動作。

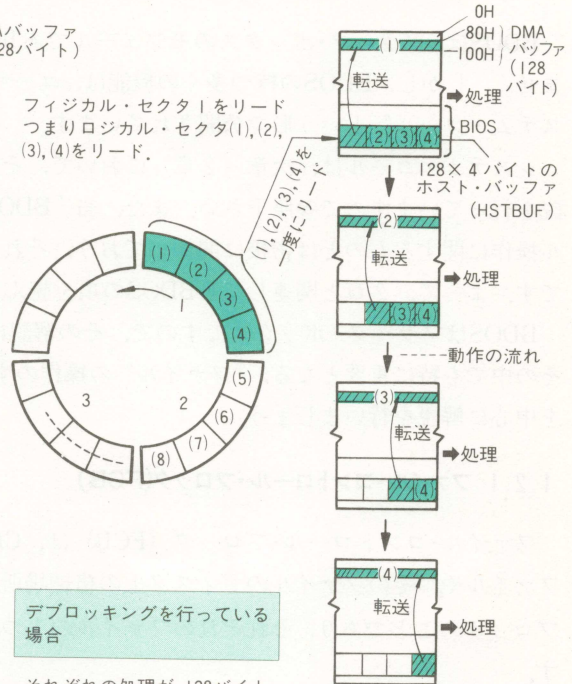
1フィジカル・セクタ=1ロジカル・セクタの場合



それぞれの処理は、128バイト単位で、DMAバッファを介して行われる。

高密度ディスク（図の例は、物理的な1セクタが512バイト）のシーケンシャルな読み出し動作。

1フィジカル・セクタ=4ロジカル・セクタの場合



それぞれの処理が、128バイト単位で、DMAバッファを介して行われることは、標準ディスクの場合と同じである。

Figure-1.1.16 デブロッキングを行う場合と、行わない場合

注) ブロッキング、デブロッキングのアルゴリズムについては、非常に専門的になりますので、デジタルリサーチ社の「CP/M 2.0 ALTERATION GUIDE」の付録Gのブロッキング、デブロッキングのアルゴリズムの骨子を示したソース・リストをご覧ください。また、このソース・ファイルは、標準CP/Mのディスクの中にも「DEBLOCK.ASM」として含まれています。ただし、この「DEBLOCK.ASM」は、1982年の前半より前のリリースのものに、最後の128バイトがディスクに書き込めないことが発生する問題点がありました。このアルゴリズムをそのまま組み込むユーザーは注意しておいて下さい。

以上、CBIOSの骨子について解説してきましたが、これらの知識を基に、CBIOSを作成あるいは変更した場合のCP/Mシステムへの組み込みと、そのシステム・ディスクの作成については、「実習CP/M」の4.10章と4.9章の、MOVCMPとSYSGENの項を参照して下さい。

1.2 BDOS(基本ディスク・オペレーティング・システム)

BDOSは、ブラック・ボックスのモジュールであり、ユーザーがそれに手を入れるということはありません。しかし、BDOSの持つ多くの機能は、ユーザーが有効に利用できるように、そのすべてが“システム・コール”という形で公開されています。

システム・コールは、次章(2章)において、そのすべての機能をサンプル・プログラム付きで解説を行っていますので参照下さい。また、当「BDOS」の項は、2章の「システム・コール」のファイル操作に関するものとは密接に関係しており、それらを別々に解説することや理解することは不可能です。よって、2章と関連して当BDOSの項を読んで下さい。

BDOSはブラック・ボックスですので、その解説は、システム・コールに関連するものになります。その中でも特に重要となる、“ファイル”の操作のすべてに関係する“FCB”(File Control Block)を中心に解説を行います。

1.2.1 ファイル・コントロール・ブロック(FCB)

ファイル・コントロール・ブロック(FCB)は、CP/Mがファイルの各種操作をするために必要な、ファイルや、そのファイルのディスク上の格納場所など、“ファイル”に関するすべての情報を持ったブロックのことであり、それぞれのファイルの1つのディレクトリに対して1つのFCBを持っています。

FCBは、ファイルがアクセスされていない定常状態では、そのファイルが存在するディスク上のディレクトリ・エリアに格納されています。今まで普通に、“ファイルのディレクトリ”と呼んでいたものは、実はディスクに格納された状態の“ファイルのFCB”であった訳です。

DISK=B																	TRACK=02					SECTOR=01																
B:00	00	4D	4F	56	43	50	4D	20	20	43	4F	4D	00	00	00	4C	.MOVCPM	COM...L																				
B:10	02	03	04	05	06	07	08	09	0A	0B	00	00	00	00	00	00																					
B:20	00	50	49	50	20	20	20	20	20	43	4F	4D	00	00	00	3A	.PIP	COM...:																				
B:30	0C	0D	0E	0F	10	11	12	13	00	00	00	00	00	00	00	00																					
B:40	00	53	55	42	4D	49	54	20	20	43	4F	4D	00	00	00	0A	.SUBMIT	COM....																				
B:50	14	15	00	00	00	00	00	00	00	00	00	00	00	00	00	00																					
B:60	00	58	53	55	42	20	20	20	20	43	4F	4D	00	00	00	06	.XSUB	COM....																				
B:70	16	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																					
.																																						
.																																						
.																																						

Figure-1.2.1 8インチ標準ディスクのディレクトリ・エリアのFCBを見る。

まず、8インチ標準ディスクのディレクトリ・エリアのFCBを実際に見てみましょう。ディレクトリ・エリアは、トラック02のセクタ01から始まっていますが(実習CP/Mの2.3章参照)、その一番最初の部分をダンプしてみます。シンクウェア・ラブズ社の“CP/MユーティリティVOL.1”の中からのセクタ・ダンプ・プログラムを使います。

このリストには、4つのファイルのFCB(通常この場合は、ディレクトリと呼ばれる)が示されています。その中のファイル“PIP.COM”について、その内容を次に示すリストで解説します。

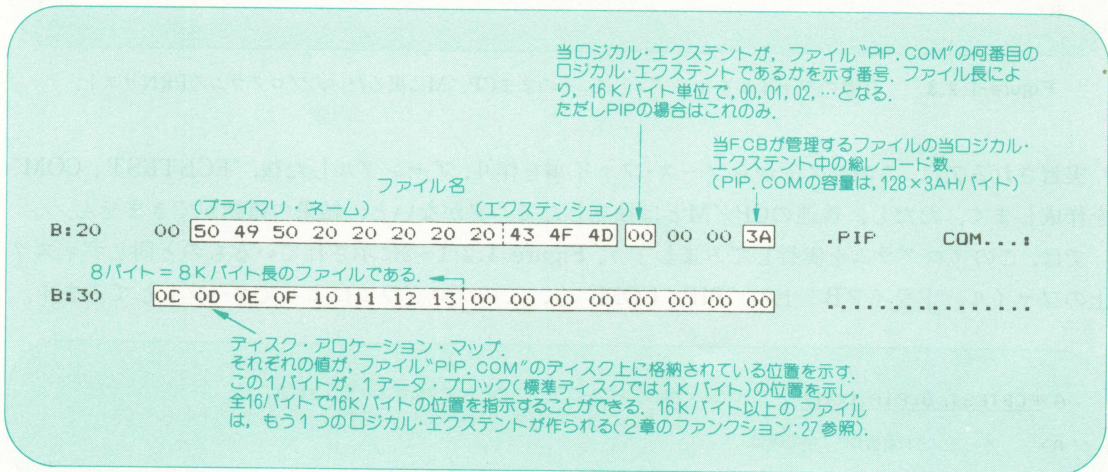


Figure-1.2.2 ディレクトリ・エリアの1つのファイルのFCBの内容:

ディスク上のディレクトリ・エリアのFCBは、それぞれのファイルに対して、このような形式で格納されています。これらのFCBは、ファイルのオープン、MAKE、リネーム、デリートなどのファンクションを行う場合、該当ファイルのものがメモリ上のFCBエリアにコピーされ、それを基にリード/ライトなどの各種の処理が行われます。メモリ上のFCBは、ファイル操作が行われるに従って更新されて行き、書き込み操作が行われた場合は、最後の“クローズ”の実行により、元のディスク上のディレクトリ・エリアに戻されます。つまりは旧FCBが更新されたことになります。

次に、ディスク上のFCBが、“ファイルのオープン”のファンクションの実行により、メモリのデフォルトのFCBエリア(5CH~7FH)にコピーされることを、実際に示してみましょう。

まず、任意のファイルをオープンして、何もせずそのままCP/Mに戻る簡単なプログラムを作ります。ファイルのオープンには、システム・コールの15番です(2章を参照下さい)。

そのプログラム“FCBTEST”のアセンブル後のPRNリストを次に示します。

A>TYPE FCBTEST.PRN!

0100	ORG	100H	
0100 0E0F	MVI	C, 15	ファイルのオープン のシステム・コール。
0102 115C00	LXI	D, 5CH	
0105 CD0500	CALL	0005H	
0108 C9	RET	CP/Mへ戻る。	
0109	END		

A>

Figure-1.2.3 任意のファイルをオープンして、そのままCP/Mに戻るだけのプログラムのPRNリスト。

実習される方は、このリストからソース・ファイルを作り、アセンブルした後、“FCBTEST.COM”を作成します。ただし、後述のCP/Mとは独立したモニタがないと、結果の確認ができません。

では、このプログラムを実行してみましょう。Figure-1.2.1~2に示されているものと同じディスク上のファイル、ドライブB:上の“PIP.COM”を、このプログラムによってオープンしてみます。

A>FCBTEST B:PIP.COM!プログラムの実行。ドライブB:上のPIP.COMがオープンされる。

A> オープンされた後CP/Mに戻る。

Figure-1.2.4 プログラムの実行。ドライブB:上のPIP.COMがオープンされる。

ディスクが数回アクセスされた後、CP/Mに戻りました。その間に、ファイル“PIP.COM”がオープンされました。このプログラムは、ファイルをオープンしたままの状態でもCP/Mに戻るため、現時点のメモリ上のFCBエリアは、ファイルがオープンされた時の状態を保っています。その時のメモリ上のFCB付近の内容を、筆者のマシンのCP/Mとは独立した別のモニタによりダンプしてみました。

注) CP/MのDDTでは、現在のFCBの内容を壊さずにダンプすることはできません。

メモリ上の現在のFCBアドレスは、デフォルトの5CH~7FHです。

このダンプリストのFCBエリアの内容と、Figure1.2.1~2のリストの内容とを比較して下さい。ディスク上のディレクトリ・エリアのFCBが、このメモリ上にコピーされているのが分かります。

デフォルトのFCBエリア.

```

0000 C3 03 BA 00 00 C3 06 AC 16 33 0E 02 06 04 79 CD .....3....y.
0010 2A 00 15 CA 00 BA 06 00 0C 79 FE 1B DA 0F 00 3E *.....y.....>
0020 53 D3 E8 DB EC 0E 01 C3 0C 00 D3 EA CD 41 00 3E S.....A.>
0030 8B B0 D3 E8 DB EC B7 F2 C3 86 A2 EB 77 23 C3 34 .....w#.4
0040 00 DB E8 E6 9D C8 1D C2 02 00 32 80 00 2F D3 FF .....2../..
0050 C3 50 00 00 00 00 00 00 00 00 00 00 02 50 49 50 .P.....PIP
0060 20 20 20 20 20 43 4F 4D 00 00 80 3A 0C 0D 0E 0F COM.....
0070 10 11 12 13 00 00 00 00 00 00 00 00 00 00 BC 43 BB .....C.
0080 0A 20 42 3A 50 49 50 2E 43 4F 4D 00 00 20 20 41 . B:PIP.COM.. A
0090 29 75 74 6F 20 6E 65 78 74 2C 20 20 20 58 29 20 )uto next, x)
00A0 61 6E 79 20 73 65 63 74 6F 72 20 20 20 52 29 65 any sector R)e
00B0 62 6F 6F 74 20 43 50 2F 4D 0D 0A 69 6E 70 75 74 boot CP/M..input
00C0 20 20 20 44 2C 20 20 4E 2C 20 20 41 2C 20 20 58 D, N, A, X
00D0 2C 20 20 52 2C 20 20 20 3E 0D 0A 0D 0A 1A 43 76 , R, >....Cv
00E0 23 D6 00 34 22 74 02 F2 02 76 23 E6 22 02 01 76 #..4"t...v#..."v
00F0 00 40 00 54 00 EC 00 56 20 60 02 6A 2F 00 12 00 .@.T...V '.j/...

```

Figure-1.2.5 ドライブB: 上のファイル "PIP . COM" がオープンされた時点のFCBエリア付近のデータ。

Figure-1.2.4に示した, 当プログラムの実行のコマンド・ライン,

A>FCBTEST B: PIP . COM

によって, なぜ "B: PIP . COM" が, このプログラムの対象ファイルとなるのかは, 後程解説します。

次に, FCBがどのように構成されているのか, そのフォーマットを図示します。アドレスは, CP/Mがデフォルトとして設定している5CH~7CHを記入してあります。

シーケンシャル・ファイルの場合は, 00~32の33バイトを使用し, ランダム・ファイルの場合は, 00~35の36バイトを使用します。

Figure-1.2.6に, ドライブB: 上のファイル "PIP.COM" がオープンされた直後のFCBを示すFigure-1.2.5を当てはめてみてください。

FCBは, 通常, アドレス5CHに置かれますが, 必要であればTPA (トランジェント・プログラム・エリア)のどこに置いてかまいません。特に複数のファイルを同時にアクセスする場合は, 2ndファイル以上のFCBは, TPAのどこかに置かざるを得ないことになります。

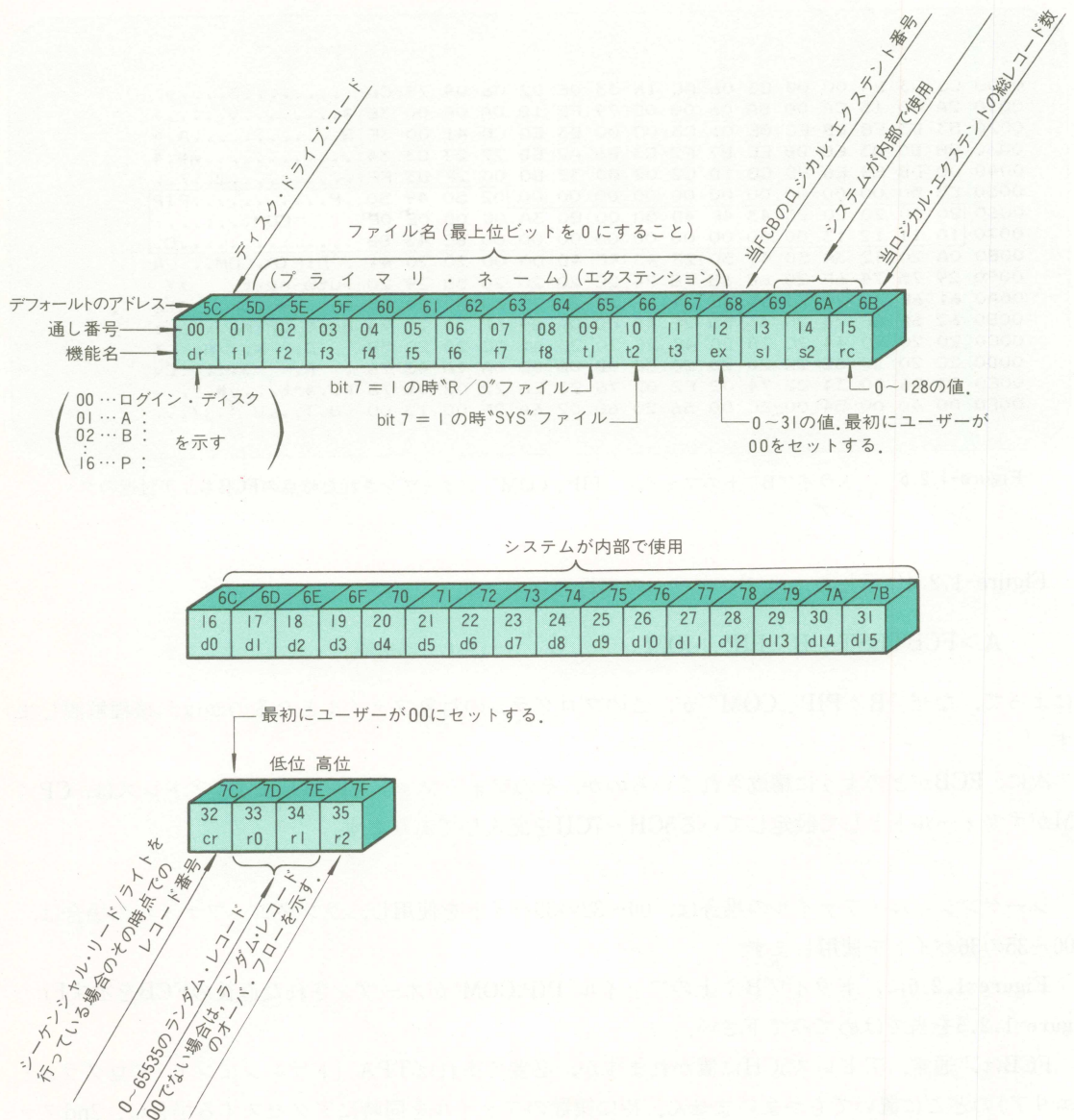


Figure-1.2.6 FCBの構成

1.2.2 プログラム実行時のコマンド・ラインとFCBの関係

Figure-1.2.4に示されているプログラムを実行する際のコマンド・ラインは、

A>x:プログラム名□x':対象となるファイル名 (x:, x':はドライブ名)

という形式をとっています。

Figure-1.2.3に示されているこのプログラムは、対象となるファイル名を、アドレス5CHからのFCBにセットしなければなりません。しかしこの仕事は、上のコマンド・ラインを実行することにより、CCPが自動的に行ってしまったのです。

この機能は、各種アプリケーション・プログラムを作成する上で、利用する場合が非常に多く、是非とも理解しておかなければならない機能の1つです。

一般的には、次のようなコマンド・ラインの形式で、2つの対象となるファイル名(1st file, 2nd file)を、5CHからのFCBにセットすることができます。

A>x:program.ext□x':1stfile.ext□x":2ndfile.ext□abcxyz.....】

このようなコマンド・ラインを実行した場合、FCBにはどのように格納されるか、実際に示してみよう。

その準備のためにまず、"RET"命令(リターン命令、コードはC9H)だけの、たった1バイトの"プログラム"を作ります。このプログラムは、プログラム自身は何の内容もなく、実行された際は、ただ、コントロールをCP/Mに戻すだけのものです。そのため、このプログラムが実行された時のコマンド・ラインは、CCPによって処理されたままの状態を保つことになります。この状態のアドレス00H~FFHのスクラッチ・エリアを、CP/Mからは独立した特別のモニターでダンプして確認しようという訳です。

RET命令1バイトだけのプログラムを作る手順を次に示します。

```
A>DDT! ..... DDTを起動
DDT VERS 2.2
-F100,1000,00! ..... 念のため、100H以後を適当に0クリアしておく。
-S100!
0100 00 C9! ..... アドレス100Hに"RET"命令(C9H)を書き込む。
0101 00 _/
-D100,10F! ..... ダンプして確認。
0100 C9 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
-^C
Ctrl-CでDDTを終わり、次はSAVEコマンドの実行。
A>SAVE 1 DUMMY.COM! ..... "RET"だけのたった1バイトの"プログラム"をセーブする。
A> ..... プログラム名は"DUMMY.COM".
```

Figure-1.2.7 "RET" 命令1バイトだけのプログラムを作る。

以上でファイル名 "DUMMY.COM" の、"何もしないことを実行する" プログラムができました。では、このプログラムを次のコマンド・ラインで実行してみましょう。

A>DUMMY B:1STFILE.ABC C:2NDFILE.XYZ OPTION

実際の実行例を次に示します。

メイン・コマンド オプション・コマンド(字数は、スペースも含めて35字=23H字)

A>DUMMY B:1STFILE.ABC C:2NDFILE.XYZ OPTION/このようなコメント・ラインを実行してみる。

A> ① ② ③ ④

Figure-1.2.8 プログラムが実行される時のコマンド・ラインと、FCBの関係を調べる実験。

このように、スクリーン上は何事もなかったようにCP/Mに戻っています。

この間にCCPは、①のプログラム・ファイル "DUMMY.COM" を、TPAにロードして100Hから実行するのはもちろんのことですが、プログラムをTPAにロードし終わり、100Hスタートで実行する直前に、コマンド・ラインの②と③が、メモリ上のFCBにセットされるのです。CCPが使用するデフォルトのFCBアドレスは、5CHからですが、それを含む0H~FFHのエリアを、Figure-1.2.8の状態のまま、筆者のCP/Mマシンに独自に設けた特別のモニターでダンプしてみましょう (DDTなどでは、FCBの状態が変化します)。

02はドライブB:) を示している。 入力されたオプション・コマンドの部分は、フォーマットに従って、FCBにコピーされている。

0000	C3	03	BA	00	00	C3	06	AC	16	33	0E	02	06	04	79	CD3.....y.
0010	2A	00	15	CA	00	BA	06	00	0C	79	FE	1B	DA	0F	00	3E	*.....y.....>
0020	53	D3	E8	DB	EC	0E	01	C3	0C	00	D3	EA	CD	41	00	3E	S.....A.>
0030	88	B0	D3	E8	DB	EC	B7	F2	41	00	DB	EB	77	23	C3	34A...w#.4
0040	00	DB	E8	E6	9D	C8	1D	C2	02	00	32	80	00	2F	D3	FF2../..
0050	C3	50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.P.....1ST
0060	46	49	4C	45	20	41	42	43	00	00	00	02	03	32	4E	44	FILE ABC.....2ND
0070	46	49	4C	45	20	58	59	5A	00	00	00	00	00	BD	D6	BA	FILE XYZ.....
0080	23	20	42	3A	31	53	54	46	49	4C	45	2E	41	42	43	20	# B:1STFILE.ABC
0090	43	3A	32	4E	44	46	49	4C	45	2E	58	59	5A	20	4F	50	C:2NDFILE.XYZ OP
00A0	54	49	4F	4E	00	00	00	00	00	00	00	00	00	00	00	00	TION.....
00B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F0	00	00	00	00	00	00	00	00	00	00	00	00	00	2F	00	12	00...../...

この値"23"は、入力されたコマンド・ラインのオプション・コマンドの字数が23Hであることを示している。

入力されたオプション・コマンドが、入力された時のままの状態で、80Hからのバツファにコピーされている(アスキー部参照)。

Figure-1.2.9 コマンド・ラインがFCBにコピーされる様子の確認。

CP/Mが使用するデフォルトのFCBアドレスは5CHです(ユーザー・プログラムでは、任意のアドレスに設定できます)。このダンプリストの5CH~7CHのFCBには、オプション・コマンド部の②と③が、Figure-1.2.6に示されているフォーマットに従って格納されていることが分かります。ただし、③については、通常はシステム内部で使用する6CHからのエリアに格納されています。

一方、④は、FCBには格納されていないことに注目して下さい。コマンド・ラインの①、②、③、④を区別するのは、それぞれの間に置いたスペースです。そして、②と③の先頭に、ドライブ名があれば、それらはFigure-1.2.6の“dr”のバイトに示されているように、ドライブ・コードに変換されてFCBに格納されます。ドライブ名がない場合は、ログイン・ディスクを指定したとみなされ、ドライブ・コード00が格納されます。

以上のように、Figure-1.2.8のコマンド・ラインを実行すると、プログラムがTPAにロードされ、100Hスタートで実行される直前に、コマンド・ラインの②と③の部分がFCBに、FCBのフォーマットで格納されることが理解されたと思います。

さらにもう1つ、重要なことがあります。

アドレス80HからのデフォルトのDMAバッファに、Figure-1.2.8のコマンドラインの②以後が、何の変更も受けずに入力した時の状態のままで格納されていることです。その先頭アドレスの80Hには、入力されたオプション・コマンド部の総入力文字が格納されます。この80Hからのエリアには、入力したものはすべて、④以後の部分も格納されるのです。

この80Hからのバッファに格納されるタイミングも、先程のFCBの場合と同じ、100Hスタートの直前です。

ユーザーは、ユーザー・プログラムの中で、前者のFCBに格納された2組のドライブ名とファイル名、それに後者のアドレス80Hからのエリアに格納された、入力されたままの状態のオプション・コマンド・ラインを、自由に利用することができます。

例えば、

```
A>DDT┘ABCD.COM┘
A>DUMP┘OPQR.XYZ┘
A>ED┘DUMP.ASM┘
```

などは前者のFCBを利用し、

```
A>MOVCPM┘64┘*┘
A>STAT┘PUN:=UP1:┘
A>PIP┘B:=ASM.COM[V]┘
```

などは後者の80Hからのバッファを利用しています。

いずれにしろ、これらのことは、次章の「システム・コール」のファイルの操作等に関するものとは密接に関係することであり、それぞれを単独で理解することは不可能です。よって、次章と本章とは一体であると考えて交互に参照して下さい。

1.3 カナ文字への対応

日本のパーソナル・コンピュータのCP/Mは、すでにカナ文字対応ができているものが多いのですが、そうでない標準CP/M上で、カナ文字の使用ができるようにするための変更箇所を示しておきましょう。これは筆者が「標準CP/Mハンドブック」にも紹介しましたが、ここではその変更作業を具体的にスクリーン上のリストで示します。

その前に、BIOS内のコンソール入出力ルーチンや、プリンタへの出力ルーチンなどには、入出力データの最上位ビットを0にする操作を行っていないということが前提です。

‘A>’のあとにカナ文字をキーインできるようにするための変更

コンソール・コマンド・レベルで、カナ文字を入力できるようにするための変更です。これを行うと、PIPコマンドの文字列サーチ・パラメータである、[S] や [Q] をカナ文字でも使用することが可能となります。また、コンソール・コマンド・レベルでなくても、EDコマンド内の文字列置き換えやサーチ・コマンドの“S”、“F”などにもカナ文字の使用が可能となります。

変更箇所は基本的には“MOVCPM.COM”をDDTでロードした時のアドレス13F5Hのデータ、“7F”が入力キャラクタの最上位ビットを0にするマスクなので、これを“FF”にでも変更して、新たにシステムを作り上げればよいのです。

実際は、次の実例に示すようにすれば簡単に新システムが作れます。CP/Mのマスター・ディスクをコピーして、それをドライブA:にセットして起動してから始めます。

```

A>12345 / .....改造前のシステム."アイウエオ"と入力したが,このように"12345"となり,
12345? .....カナとしては受付けていない.

A>SYSGEN / .....SYSGENを起動.
SYSGEN VER 2.2
SOURCE DRIVE NAME (OR RETURN TO SKIP) A } ドライブA: 上のCP/Mシステムをメモリ上にロードする.
SOURCE ON A, THEN TYPE RETURN /
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) / .....メモリ上にロードしたままCP/Mに戻る.

      ↓ このページ数は,それぞれのCP/MのBIOSの大きさにより異なる. オープするのはかまわない.
A>SAVE 40 CPM.COM / .....メモリ上のCP/Mシステムのイメージを,ファイル"CPM.COM"として, ディスクにセーブする.

A>DDT CPM.COM / .....DDTを起動し, 先程の"CPM.COM"をロードする.
DDT VERS 2.2
NEXT PC
2900 0100
-D13F0,13FF / .....確認のためのダンプ.
13F0 E5 CD FB CC E6 7F E1 C1 FE OD CA C1 CE FE OA CA .....この"ANI 7FH"により,入力キャラクタの最上位bitが0にされる.
-S13F5 /
13F5 7F FF / ..... "7F"を"FF"にでも変更する(すべてのキャラクタが通る).
13F6 E1 / .....
-GO / .....リブートしてCP/Mに戻る.改造されたCP/Mシステムはメモリ上にある.

A>SYSGEN /
SYSGEN VER 2.2 .....SYSGENを起動.
SOURCE DRIVE NAME (OR RETURN TO SKIP) / .....スキップする.
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) A } メモリ上の改造されたCP/Mシステムを,
DESTINATION ON A, THEN TYPE RETURN / .....ドライブA: のシステム・トラックに組み込む.
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) / .....改造システム・ディスクで書き上り,CP/Mに戻る.

A>

```

Figure-1.3.1 カナ対応のCP/Mシステムを作成する手順.

リセット

```

A>アイウエオ / .....改造されたCP/Mを起動した後,"アイウエオ"と入力してみた.
アイウエオ? .....このように受付られている.

A>

```

Figure-1.3.2 コンソール・コマンドのレベルでもカナが使えます.

DDTのダンプ・コマンドのアスキー表示部にカナ表示をさせるための変更

これもCP/Mのマスター・ディスクからコピーした“DDT.COM”を準備して、次の手順で行います。

```

A>DDT DDT.COM / .....DDTを起動し、自分自身をロードする。
DDT VERS 2.2
NEXT PC .....確認のためのダンプ。
1400 0100
-DE30,E3F / .....この“CPI 7FH”でその値以上のキャラクタを“.”にしてしまう。
0E30 05 0C 7D C3 05 0C FE 7F D2 40 0C FE 20 D2 C7 0B ...}.....@... ..カナは表示され
-SE37 / .....ない。
0E37 7F FF .....それを“FF”とすればすべてのキャラクタが、そのままコンソールに送られ表示される。
0E3B D2 _ /
-GO / .....リポートしてCP/Mに戻る

A>SAVE 19 DDT.COM / .....メモリ上の改造DDTをセーブする。改造DDTのでき上り。

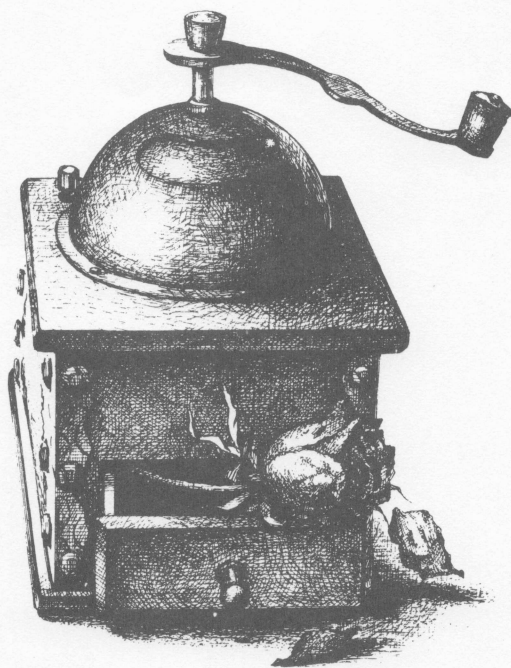
A>DDT DDT.COM / .....改造DDTをセーブする。
DDT VERS 2.2
NEXT PC
1400 0100
-DE30,E4F /
0E30 05 0C 7D C3 05 0C FE FF D2 40 0C FE 20 D2 C7 0B ...}チ...メ@. メヌ.
0E40 3E 2E C3 C7 0B EB 2A 5F 0F 7D 93 6F 7C 9A EB C9 >.チヌ.*_.}ロ! / カナが表示されている。
-GO /

A>

```

Figure-1.3.3 アスキー表示部にカナを表示させるためのDDTの改造。

2章 全システム・コール徹底解説



2.1 システム・コールとは

システム・コールとは、CP/Mが持っているコンソールなどの各周辺装置との入出力機能や、ディスク・ファイルの各種操作などの機能を、外部のユーザー・プログラムから使用することを許したCP/Mの“システム・サービス”のことです。

システム・コールは、“ファンクション・コール”や、“スーパーバイザ・コール”とも呼ばれており、CP/Mにとってこの機能は大変重要なものであり、システム・コールあつてのCP/Mと言っても良いでしょう。

CP/Mが、今日、マイクロコンピュータのOSの標準になり得たのも、この多くの機能を持つシステム・コールを、使いやすい形式に整備して、ユーザーに提供した点にあると筆者は考えています。

ハードウェアに依存しない、このシステム・コールを利用して、各ソフトウェア・メーカーは、いっせいに各種のソフトウェアを製品化しました。つまり、システム・コールを使ったこれらのソフトウェアは、CP/Mマシンならば、すべての機種で実行することができるのです。そして、これらのソフトウェアが、今日CP/Mを取り巻く、おびただしい数のソフトウェア群となっているのです。

さて、みなさんが、CP/M上で実行する何らかのプログラムを、アセンブラで作る場合を想定して下さい。まず、コンピュータの入出力で最も基本的な、コンソールの入出力について考えてみましょう。

「キーボードから1文字入力するプログラムと、スクリーンへ1文字出力させるプログラムを作りなさい。」という問題です。

さて、あなたならどうしますか？

エーと、このコンピュータのキーボード・データの入力ポート・アドレスは、確かxxHで、ステータスbitはbit0で、……と考え始めた方は、どうしてもこの章を読まなければなりません。こんなに便利な機能があるのです。

問題のプログラムは、システム・コールを使うと、次のようにスマートに解決します。

コンソール入力（通常はキーボードから1文字入力）

```
MV 1    C, 1  
CALL 0005H
```

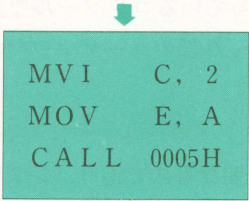


入力があれば、入力データがAレジスタに入っている

一方、コンソールへ1文字出力する場合は次のようになります。

コンソール出力（通常はスクリーンに1文字出力）

出力データをAレジスタに持っている場合、



```
MVI    C, 2
MOV    E, A
CALL   0005H
```

Aレジスタの文字が、コンソールに出力される。

このようにわずか2～3ステップでこれらの機能が記述できるのです。そして、重要なことは、ここには、先程のポート・アドレスのような、コンピュータのハードウェアに関することが、表面には一切現れていないことです。従って機種の変ななどのようなCP/Mマシンでも、このプログラムは共通に動作することになります。

システム・コールの一般形を次に示します。

システム・コールの一般形

```
MVI      C, function No.
(LXI     D, address あるいは MVI E, parameter)
CALL     0005H
```

（結果を返すものは、結果がAあるいは（H，L）レジスタに格納される。）

つまり、

- ★ファンクションNo.を“Cレジスタ”にセット。
- ★（ファンクションによっては、パラメータを（D，E）ペアレジスタまたはEレジスタにセット。）
- ★アドレス0005番地をCALL。

この、2ステップまたは3ステップで、Figure-2.1.1のシステム・コール一覧表にある、すべての機能が実現できるのです。

CALLした結果は、コンソールへの1文字出力のように、直接に動作や現象として現れるファンクションのものと、結果のデータが得られるファンクションでは、1バイトの場合Aレジスタに、2バイトの場合(H, L)ペアレジスタにデータがセットされて戻るものがあります(CP/M ver1.4とコンパチビリティを保つため、H, Lレジスタと同じ値が、A, Bレジスタにも格納されています)。つまり、結果は、

★動作として現れる。

★1バイトの結果はAレジスタに、2バイトの結果は(H, L)ペアレジスタに格納されている。

この機能が、CP/Mの“システム・コール”と呼ばれるものです。

次頁にシステム・コールのファンクション一覧表を示します。

システム・コールにおけるスタックの扱いについて

システム・コールされた内部では、独自のスタック・ポインタが使用されており、ユーザー・プログラム上のスタックは一切消費しません。よってユーザーは、システム・コール内部でのスタックの消費を考慮する必要はありません。

ここでのシステム・コールのサンプル・プログラムは、簡素化のために、ほとんどのものはユーザー・プログラムでスタック・ポインタを設定せずに、CCP内のスタックをそのまま継続して使っています。CCPからユーザー・プログラムにコントロールが移った時に、CCPが使用していた7レベルのスタックが使用可能ですが、これ以上にスタックが深くなるプログラムの場合は、ユーザー・プログラムで、新たにスタック・ポインタを設定する必要がありますので注意して下さい。

ファンクションNo. (Cレジスタにセットする) 10進 HEX		ファンクション	パラメータのセット (DEレジスタまたはEレジスタへ)	結果 (DEレジスタまたはEレジスタへ)	
周辺装置との入出力	0	00	システム・リセット	なし	システムがリセットされる
	1	01	CON: (コンソール)からの入力	なし	A←入力キャラクタ
	2	02	CON: (コンソール)への出力	E←キャラクタ	CON:へ出力される
	3	03	RDR: (リーダー)からの入力	なし	A←入力キャラクタ
	4	04	PUN: (パンチ)への出力	E←キャラクタ	PUN:へ出力される
	5	05	LST: (リスト)への出力	E←キャラクタ	LST:へ出力される
	6	06	ダイレクト・コンソール入出力	入力: E←FFH 出力: E←キャラクタ	入力: A←キャラクタ 出力: コンソールへ出力
	7	07	IOバイトの取り出し	なし	A←IOバイト
	8	08	IOバイトのセット	E←IOバイト	IOバイトがセットされる
	9	09	文字列のプリントアウト	DE←文字列アドレス	\$記号までの文字列がコンソールに出力
	10	0A	コンソール・バッファへの読み込み	DE←バッファ・アドレス	コンソールからバッファへ入力
バージョン1.4 の使用可能範囲	11	0B	CON: (コンソール)入力ステータスのチェック	なし	入力あり: A←FFH 入力なし: A←00
	12	0C	バージョンNo.の取り出し	なし	H←CP/M-MP/M L←バージョンNo.
	13	0D	ディスク・システムのリセット	なし	すべてのディスクがリセットされる
	14	0E	ディスク・ドライブのセレクト	E←ドライブNo.	デフォルト・ドライブに指定される
	15	0F	ファイルのオープン	DE←FCBアドレス	正常: A←ディレクトリ・コード ファイルがない: A←FFH
	16	10	ファイルのクローズ	DE←FCBアドレス	
	17	11	最初のファイルのサーチ	DE←FCBアドレス	
	18	12	次のファイルのサーチ	なし	
	19	13	ファイルのデリート	DE←FCBアドレス	正常: A←00 終了またはディスク・フル: A←00以外 正常 A←ディレクトリ・コード ディレクトリ・フル: A←FFH 正常 A←ディレクトリ・コード ディレクトリ・フル: A←FFH ファイルがない: A←FFH
	20	14	シーケンシャル・リード	DE←FCBアドレス	
	21	15	シーケンシャル・ライト	DE←FCBアドレス	
	22	16	ファイルの作成	DE←FCBアドレス	
	23	17	ファイル名の変更	DE←FCBアドレス	正常 A←ディレクトリ・コード ディレクトリ・フル: A←FFH 正常 A←ディレクトリ・コード ディレクトリ・フル: A←FFH ファイルがない: A←FFH
	24	18	ログイン・ベクトルの取り出し	なし	
25	19	ログイン・ディスクNo.の取り出し	なし	A←ディスクNo.	
26	1A	DMAアドレスのセット	DE←DMAアドレス	DMAアドレスがセットされる	
ディスク入出力関係	27	1B	アロケーション・アドレスの取り出し	なし	HL←アロケーション・ベクトル・アドレス
	28	1C	ライト・プロテクトのセット	なし	ログイン・ディスクがR/Oにセットされる
	29	1D	R/Oベクトルの取り出し	なし	HL←ベクトル
	30	1E	ファイル・アトリビュートのセット	DE←FCBアドレス	A←ディレクトリ・コード ファイルがない: A←FFH
	31	1F	ディスク・パラメータ・アドレスの取り出し	なし	HL←DPBのベース・アドレス
	32	20	ユーザー・コードのセット/取り出し	セット: E←ユーザー・コード 取り出し: E←FFH	ユーザー・エリアが変更される A←ユーザー・コード
	33	21	ランダム・リード	DE←FCBアドレス	正常: A←00
	34	22	ランダム・ライト	DE←FCBアドレス	エラー: A←エラーコード
	35	23	ファイル・サイズの計算	DE←FCBアドレス	FCBのr0-r2←ファイル・サイズ
	36	24	ランダム・レコードのセット	DE←FCBアドレス	FCBのr0-r2←ランダム・レコードNo.
	37	25	ディスク・ドライブのリセット	DE←ドライブ・ベクトル	ベクトルのドライブがリセットされる
	バージョン2.0以上 使用可能	38	26	使用されていない	
		39	27		
		40	28	ゼロ・フィルを伴うランダム・ライト	DE←FCBアドレス

注) HLレジスタと同じ内容がA Bレジスタにもセットされる。

注) ファンクション12は、バージョン1.4では「ヘッドのリフトアップ」のファンクションである。

注) ディレクトリ・コード: 128バイト中の該当ディレクトリの位置により、0~3の値をとる。

Figure-2.1.1 システム・コール一覧表

2.2 システム・コール徹底実習

何はともあれ、これらのすべての機能を簡単なプログラムで実習してみましょう。実習用のサンプル・プログラムは、それぞれの“システム・コール”を、やさしく解説することを第1目的とした(アルゴリズムのスマートさなどは、問題にしているわけではありませんのであしからず)。それぞれのプログラム・リストとそのコメント文が、システム・コールについての何よりの解説になりますので、リストをよく見て下さい。

最初に示す実習プログラム Figure-2.2.1だけをみても、システム・コールの概念はつかめると思います。

ファンクション：0,1,2の実習

ファンクション：0…システム・リセット

CALL手順

```
[ MVI    C, 0 ]
[ CALL   0005H ]
```

機能

0番地へのジャンプ、あるいはCtrl-Cのキーインを行ってリブートした場合と同じ機能である。このファンクションを実行すると、“WBOOT”（BIOS+3番地）へのジャンプが行われる。

ファンクション：1…コンソールからの入力

CALL手順

```
[ MVI    C, 1 ]
[ CALL   0005H ]
  ↓
[ (入力キャラクタがAレジスタに格納されている) ]
```

機能

コンソール(通常はキーボード)からの入力があれば、入力キャラクタをAレジスタに持ってCALLから戻る。と同時に、コンソール(通常はスクリーン)に入力キャラクタを出力(エコー・バック)する。入力がない場合は、入力されるまで内部でループしている。

コンソールへのエコー・バックに関しては、Ctrl-S、Ctrl-P、タブの各機能が働く（ファンクション：2を参照）。

ファンクション：2…コンソールへの出力

CALL手順

```
[ MVI    C, 2
  (E) ← 出力キャラクタ
  CALL   0005H ]
```

機能

EレジスタにセットされたASCIIキャラクタを、コンソール（通常はスクリーン）に出力する。この出力に関しては、コンソールからCtrl-Sによるポーズや、事前のCtrl-Pによるリスト出力、さらに8文字ごとのタブ処理がそれぞれ機能する。

実習プログラム ファンクション：0,1,2

コンソールから1文字入力すると、コロン“:”に続いて入力された文字が、255文字連続してコンソールに出力され、Ctrl-Rを入力するとシステム・リセットが行われる。

このようなプログラムを作ってみましょう。
このプログラムのPRN形式のソース・リストを次に示します。

```
A>TYPE 0-1-2.PRN/

;-----;
;  FUNCTION  0: SYSTEM RESET
;             1: CONSOLE INPUT
;             2: CONSOLE OUTPUT
;-----;

0100          ORG      100H
START:
0100 0E02      MVI      C,2
0102 1E2A      MVI      E,'*'
0104 CD0500    CALL     0005H    } "コンソール出力"のシステム・コール、
                                } "*"がスクリーンに出力される。

0107 0E01      MVI      C,1
0109 CD0500    CALL     0005H    } "コンソール入力"のシステム・コール、
                                } キー入力があるまで、この内部でループして
                                } いる。

010C 324901    STA      BUFF
010F FE12      CPI      12H ; =^R
0111 CA4401    JZ       RESET    } キー入力データが"BUFF"へストア、
                                } キー入力データがCtrl-Rのときは、"RESET"
                                } ヘジャンプ。

;-----;
```


0114 0E02	MVI	C, 2	} "コンソール出力" のシステム・コール。 ": "がスクリーンに出力される。
0116 1E3A	MVI	E, ':",	
0118 CD0500	CALL	0005H	
011B 06FF	MVI	B, 255 出力文字数のカウンタ. 255文字出力するため.
011D 3A4901	LDA	BUFF キー入力データをAレジスタにロード.
0120 C5	PUSH	B	
0121 0E02	MVI	C, 2	} "コンソール出力" のシステム・コール。 Aレジスタの文字がスクリーンに出力される。
0123 5F	MOV	E, A	
0124 CD0500	CALL	0005H	
0127 C1	POP	B	
0128 05	DCR	B	} 255文字出力するまでループする.
0129 C21D01	JNZ	LOOP	
012C 0E02	MVI	C, 2	} キャリッジ・リターンをスクリーンに出力.
012E 1E0D	MVI	E, 0DH	
0130 CD0500	CALL	0005H	
0133 0E02	MVI	C, 2	} ライン・フィードをスクリーンに出力.
0135 1E0A	MVI	E, 0AH	
0137 CD0500	CALL	0005H	
013A 0E02	MVI	C, 2	} もう1度ライン・フィードを スクリーンに出力.
013C 1E0A	MVI	E, 0AH	
013E CD0500	CALL	0005H	
0141 C30001	JMP	START 最初から繰り返し.
0144 0E00	MVI	C, 0	} "システム・リセット" のシステム・コール。 リポートと同じ機能なので、このあとに続く プログラムはない。
0146 CD0500	CALL	0005H	
0149	DS	1 キー入力の1文字バッファ.
014A	END		

A> 注) コンソール入出力を、キー入力、スクリーン出力として説明しています。

Figure-2.2.1 ファンクション：0, 1, 2 実習プログラムのPRN形式ソース・リスト。

上記ソース・リストから、エディタを使ってアセンブリ・ソース・ファイルを作り、ASM→LOADを行い、"0-1-2.COM" を生成し実行してみましょう。

ファンクション：0, 1, 2 実習プログラムの実行

"0-1-2.COM" を実行すると、プロンプト "*" が出力されるので、適当な文字をキーインします。
": " に続いて、入力された文字が255文字出力されます。

文字のキーイン直後に素早く Ctrl-S をキーインして、255文字の出力がポーズになることや、事前の Ctrl-P によるプリンタへの出力、タブ (TAB キーがない場合は Ctrl-I で代用できる) の処理などが機能することを確認して下さい。

実行例を次に示します。

A>0-1-2/

“コンソール入力” 自身の表示。

← 入力された文字が255文字 “コンソール出力” されている。

```
*A: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAA   キャリッジ・リターンとライン・フィードが行われている。
```

← もう1度ライン・フィードが行われている。

```
*B: 5555555555555555555555555555555555555555555555555555555555555555
5555555555555555555555555555555555555555555555555555555555555555
5555555555555555555555555555555555555555555555555555555555555555
555555555555555555
```

```
*Z: 7777777777777777777777777777777777777777777777777777777777777777
7777777777777777777777777777777777777777777777777777777777777777
7777777777777777777777777777777777777777777777777777777777777777
777777777777777777
```

*^R ← Ctrl-Rの入力により, “システム・リセット” が行われ, CP/Mに戻った。

A>

注) Ctrl-S, Ctrl-P, Ctrl-Iなどの実験も行って下さい。

Figure-2.2.2 ファンクション：0, 1, 2 実習プログラムの実行。

ファンクション：3,4,5の実習

ファンクション：3…リーダからの入力

CALL手順

MVI	C, 3
CALL	0005H
↓	
(入力データはAレジスタに格納されている)	

機能

ロジカル・デバイスの “RDR:” に入力があれば, 入力データをAレジスタに持って, CALLから戻る。入力がない場合は, 入力されるまで内部でループしている。

ファンクション：4…パンチへの出力

CALL手順

MVI	C, 4
(E) ←	出力データ
CALL	0005H

機能

Eレジスタにセットされているデータを，ロジカル・デバイスの“PUN:”に出力する。

ファンクション：5…リストへの出力

CALL手順

MVI	C, 5
(E) ←	出力データ
CALL	0005H

機能

Eレジスタにセットされているデータを，ロジカル・デバイスの“LST:”に出力する。

実習プログラム ファンクション：3,4,5

“RDR:”デバイスから入力したデータを，メモリ上にバッファして行き，Ctrl-P (10H) が入力されると，今までにバッファした内容を“PUN:”デバイスへ出力する。また，Ctrl-L (0CH) が入力されると，“LST:”デバイスに出力する。Ctrl-Z (1AH) が入力されると，本プログラムを終了し，CP/Mに戻る。

このようなプログラムを作ってみましょう。

このプログラムのPRN形式のソース・リストを示します。

A>TYPE 3-4-5.PRN

```

;-----;
;  FUNCTION  3: READER INPUT      ;
;  FUNCTION  4: PUNCH OUTPUT      ;
;  FUNCTION  5: LIST OUTPUT        ;
;-----;

0100          ORG      100H
0100 214B01  START:  LXI      H, BUFF ..... H, Lレジスタに“BUFF”のアドレスをセット。

```


0103 E5	PUSH	H	} リーダ入力"のシステム・コール。 アドレス・ポインタであるH, Lレジスタを 保護している。
0104 0E03	MVI	C, 3	
0106 CD0500	CALL	0005H	
0109 E1	POP	H	
010A FE10	CPI	10H ; ^=P	} 入力がCtrl-Pならば、バッファされた内容を パンチ出力へ。 入力がCtrl-Lならば、バッファされた内容を リスト出力へ。 入力がCtrl-Zならば、本プログラムを終了し、 CP/Mへ。
010C CA1E01	JZ	PUNOUT	
010F FE0C	CPI	0CH ; ^=L	
0111 CA3301	JZ	LSTOUT	
0114 FE1A	CPI	1AH ; ^=Z	} リーダ入力データのバッファリング。 最終データの次に、必ず"00"をストアする。 ----- 次のデータ入力ヘループ。
0116 CB	RZ		
0117 77	MOV	M, A	
0118 23	INX	H	
0119 3600	MVI	M, 0	
011B C30301	JMP	START+3	
PUNOUT:			
011E 214801	LXI	H, BUFF	} バッファから出力データの取り出し。 データが"00"の場合"START"へのジャンプ。 繰り返し。
0121 7E	MOV	A, M	
0122 FE00	CPI	0	
0124 CA0001	JZ	START	
0127 E5	PUSH	H	} "パンチ出力"のシステム・コール。 H, Lレジスタを保護している。
0128 0E04	MVI	C, 4	
012A 5F	MOV	E, A	
012B CD0500	CALL	0005H	
012E E1	POP	H	
012F 23	INX	H	} 次のデータ出力ヘループ。
0130 C32101	JMP	PUNOUT+3	
LSTOUT:			
0133 214801	LXI	H, BUFF	} バッファから出力データの取り出し。 データが"00"の場合"START"へジャンプ。 繰り返し。
0136 7E	MOV	A, M	
0137 FE00	CPI	0	
0139 CA0001	JZ	START	
013C E5	PUSH	H	} "リスト出力"のシステム・コール。 H, Lレジスタを保護している。
013D 0E05	MVI	C, 5	
013F 5F	MOV	E, A	
0140 CD0500	CALL	0005H	
0143 E1	POP	H	
0144 23	INX	H	} 次のデータ出力ヘループ。
0145 C33601	JMP	LSTOUT+3	
0148 =	BUFF	EQU	\$ ----- 入力データのバッファの始まり。
0148	END		

A>

A>

Figure-2.2.3 ファンクション：3，4，5 実習プログラムのPRN形式ソース・リスト。

このソース・リストからアセンブリ・ソース・ファイルを作り、"3-4-5.COM"を生成してプログラムを実行してみましょう。

ファンクション：3, 4, 5 実習プログラムの実行

プログラム "3-4-5. COM" を起動させ外部から "PUN：" デバイスへデータを送り込みます。本書の場合は、もう 1 台の CP/M マシンから、互いの RS-232C インターフェイス ("PUN："、"RDR：" 両デバイスに割り当てられている) を介してデータを送っています。

Figure-2.2.4 は、データ送り出し側の実行例であり、PIP により、キーボード入力を "PUN：" デバイスから出力しています。実習プログラム "3-4-5. COM" が起動している受信側では、この出力を "RDR：" デバイスから入力して、各動作が行われる訳です。

Figure-2.2.5 は、実習プログラムを実行した時のスクリーンの表示例であり、Figure-2.2.6 は、"LST：" デバイスに出力された印字例です。

```
A>PIP PUN:=CON:J ... "CON：" (通常はキーボード) 入力を "PUN：" から出力.
```

```

ABCDEFGHIJKLMN / LF
0123456789 / LF
abcdefghijklmn / LF
^L ..... Ctrl-L の入力により、上記データを "LST：" へ出力させる.
OPQRSTUVWXYZ / LF
9876543210 / LF
opqrstuvwxyz / LF
^P ^Z ..... Ctrl-P の入力により、上記データを "PUN：" へ出力させる.
A> ..... Ctrl-Z の入力により、プログラム "3-4-5" を終了させ、
          こちらの PIP も終了させる.
```

注) LF はライン・フィード・キーの入力を表す。LF は Ctrl-J で代用できる。

Figure-2.2.4 この実習でのデータ送り出し側 CP/M マシンの実行例。

```
A>3-4-5J
```

```
A>..... Ctrl-Z が入力されたので、CP/M に戻った.
```

Figure-2.2.5 ファンクション：3, 4, 5 実習プログラムの実行例。Figure-2.2.4 でのデータが入力される。

```

ABCDEFGHIJKLMN
0123456789
abcdefghijklmn
```

Figure-2.2.6 ファンクション：3, 4, 5 実習プログラムによる "LST：" デバイスへの出力例。

ファンクション：6の実習

ファンクション：6…ダイレクト・コンソール入出力

CALL手順

```

    入力の場合
    MVI    C, 6
    MVI    E, 0FFH
    CALL   0005H
    ↓
    (入力データはAレジスタに格納されている)
  
```

```

    出力の場合
    MVI    C, 6
    (E) ← 出力データ
    CALL   0005H
  
```

機能

EレジスタにFFHをセットしてCALLする場合はコンソール入力、FFH以外であればコンソール出力となる。入力の場合は、入力があれば入力データをAレジスタに持って、CALLから戻る。入力がない場合は、Aレジスタに00を持ってCALLから戻る（入力があるまで待たないことに注目）。出力の場合は、Eレジスタにセットされているデータを、コンソールに出力する。いずれの場合も、ファンクション1および2で有効であったCtrl-S、Ctrl-Pなどは機能しない。

実習プログラム ファンクション：6

コンソールへ“ ”記号を連続して出力させながら、コンソール入力を監視し、入力があれば、そのキャラクタをコンソールへ出力する。入力がCtrl-Zであれば本プログラムを終了してCP/Mに戻る。

このようなプログラムを作ってみましょう。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 6.PRN /

```

;-----;
;  FUNCTION  6: DIRECT CONSOLE I/O  ;
;-----;

0100          ORG      100H
START:
0100 0E06      MVI     C,6
0102 1EFF      MVI     E,0FFH
0104 CD0500     CALL    0005H
; "ダイレクト・コンソール入出力"の"入力"の
; システム・コール。EレジスタにFFHをセット
; していることに注目。入力が無い場合、内部で
; ループしないことに注目。
0107 B7        ORA     A
0108 C21501     JNZ     CHROUT
; 入力(通常はキー入力)があったかどうかの判
; 定。入力があつた場合、"CHROUT"にジャン
; プ。
010B 0E06      MVI     C,6
010D 1E2D      MVI     E,'-'
010F CD0500     CALL    0005H
; "ダイレクト・コンソール入出力"の"出力"の
; システム・コール。 '-'記号をコンソールに
; 出力する(通常はスクリーンに出力)。
0112 C30001     JMP     START  ; 最初に戻ってループ。

CHROUT:
0115 FE1A      CPI     1AH
0117 C8        RZ
; 入力がCtrl-Zなら本プログラムを終了し、
; CP/Mへ戻る。
011B 0E06      MVI     C,6
011A 5F        MOV     E,A
011B CD0500     CALL    0005H
; "ダイレクト・コンソール入出力"の"出力"の
; システム・コール。入力キャラクタをコンソ
; ールに出力する。
011E C30001     JMP     START  ; 最初に戻ってループ。

0121          END

```

A>

Figure-2.2.7 ファンクション：6 実習プログラムのPRN形式ソース・リスト。

このソース・リストからアセンブリ・ソース・ファイルを作り、"6.COM" を生成してプログラムを実行してみましょう。

ファンクション：6 実習プログラムの実行

プログラム "6.COM" を起動すると、スクリーンに "-" 記号が連続して出力されますので、適当にキーインを行うと、キーインされたキャラクタが表示されて行きます。このプログラムは、キー入力を監視しながら、スクリーンへの出力を連続して行っている訳です。Ctrl-S, Ctrl-Pの機能が働かないことも確認して下さい。Ctrl-Zを入力すると、プログラムを終了してCP/Mへ戻ります。

この実行例を次に示します。

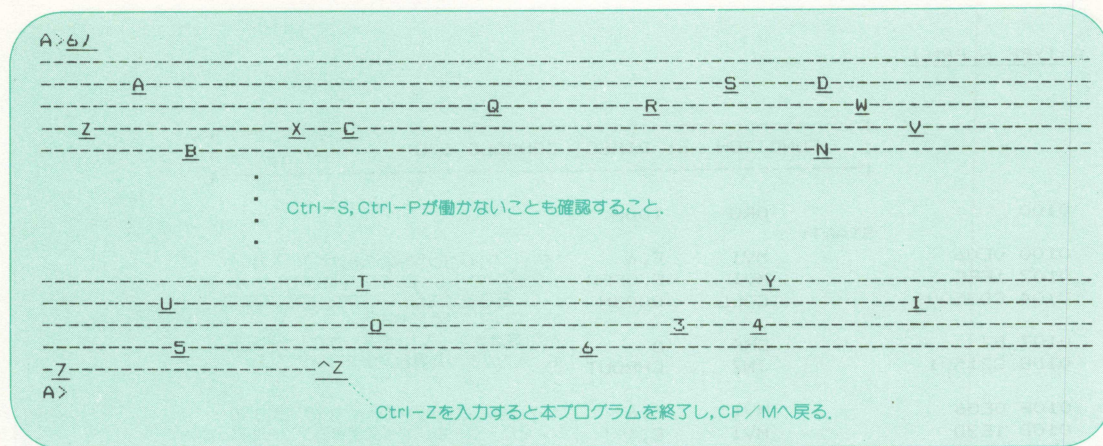


Figure-2.2.8 ファンクション：6 実習プログラムの実行.

ファンクション：7,8の実習

ファンクション：7…IOバイトの取り出し

CALL手順

```

[ MVI    C, 7
  CALL    0005H
  ↓
  (IOバイトの値がAレジスタに格納されている) ]
    
```

機能

アドレス0003HのIOバイトの値を、Aレジスタに持ってCALLから戻る。

ファンクション：8…IOバイトのセット

CALL手順

```

[ MVI    C, 8
  (E) ←  IOバイトの値
  CALL    0005H ]
    
```

機能

Eレジスタの値が、アドレス0003HのIOバイトにセットされる。

実習プログラム ファンクション：7,8

プログラム実行前のIOバイトの値をコンソールに表示し、次に、ロジカル・デバイスの“RDR:”と“PUN:”に、それぞれフィジカル・デバイスの“UR1:”, “UP1:”を割り当て、割り当てた後のIOバイトの値を、再びコンソールに表示する。

このようなプログラムを作ってみましょう。

このプログラムのPRN形式のソース・リスト次に示します。

A>TYPE 7-8.PRN/

```

;-----;
;      FUNCTION  7: GET I/O BYTE      ;
;      8: SET I/O BYTE                ;
;-----;
0100          ORG      100H
START:
0100 CD4501    CALL    CRLFOUT ..... 復帰・改行をコンソールへ出力するサブルーチン.
0103 0E07      MVI     C, 7           } "IO/バイトの取り出し"のシステム・コール.
0105 CD0500    CALL    0005H         } 現在のIO/バイトの値がAレジスタに得られる.

0108 F5        PUSH    PSW           } 得られた(A)レジスタの値を16進でコンソール
0109 CD2601    CALL    HEXDSPLY      } に出力するサブルーチン.
010C F1        POP     PSW           } (A)レジスタを保護している.

010D E6C3      ANI     11$00$00$11B } IO/バイト(Aレジスタ)の“RDR:”と
010F F62B      ORI     00$10$10$00B } “PUN:”のビットを0クリアし、そこ
0111 5F        MOV     E, A          } にUR1:とUP1:をセットする.
0112 0E08      MVI     C, 8          } CON.とLST:は変化させない.
0114 CD0500    CALL    0005H         } “IO/バイトのセット”のシステム・コール.
                                } 上記(A)レジスタの値がIO/バイトにセットされる.

0117 CD4501    CALL    CRLFOUT ..... 復帰・改行をコンソールへ出力.

011A 0E07      MVI     C, 7           } “IO/バイトの取り出し”のシステム・コール.
011C CD0500    CALL    0005H         } 変更したIO/バイトを取り出して再表示させるため.

011F CD2601    CALL    HEXDSPLY ..... 変更されたIO/バイトの値を16進でコンソールに出力.
0122 CD4501    CALL    CRLFOUT ..... 復帰・改行を出力.

0125 C9        RET ..... 本プログラムを終了し、CP/Mに戻る.

HEXDSPLY:
0126 F5        PUSH    PSW
0127 0F0F0F0F  RRC! RRC! RRC! RRC
012B CD2F01    CALL    HOUT1
012E F1        POP     PSW
012F E60F      HOUT1:  ANI     0FH           } (A)レジスタの値を16進でコンソールに表示す
0131 C630      ADI     30H           } るサブルーチン.
0133 FE3A      CPI     3AH
0135 DA3A01    JC      HOUT2
0138 C607      ADI     7
013A CD3E01    HOUT2:  CALL    CONOUT
013D C9        RET

```


013E 0E02	CONOUT:	MVI	C,2	"ファンクション2"のコンソール出力のサブ ルーチン。(A)レジスタのキャラクタが、コン ソールに出力される。
0140 5F		MOV	E,A	
0141 CD0500		CALL	0005H	
0144 C9		RET		
CRLFOUT:				
0145 3E0D		MVI	A,0DH	復帰・改行をコンソールに出力するサブルーチン。
0147 CD3E01		CALL	CONOUT	
014A 3E0A		MVI	A,0AH	
014C CD3E01		CALL	CONOUT	
014F C9		RET		
0150	END			

A>

Figure-2.2.9 ファンクション：7，8 実習プログラムのPRN形式ソース・リスト

このソース・リストからアセンブリ・ソース・ファイルを作り，"7-8.COM" を生成してプログラムを実行してみましょう。

ファンクション：7,8 実習プログラムの実行

まず最初に，STATコマンドで，現在のフィジカル・デバイスのアサイン（割り当て）状況を見ておきましょう。この実行例を次に示します。

A>STAT DEV: /	-----STATコマンドでフィジカル・デバイスの初期のアサイン状況を見る。
CON: is UC1:	
RDR: is TTY:	"TTY:"であることを注目。
PUN: is TTY:	
LST: is LPT:	
A>	

Figure-2.2.10 実習プログラム実行前の各デバイスのアサイン状況を見る。

ついでに，DDTコマンドで，IOバイト付近をダンプしておきます。

A>DDT /	-----DDTによる実行前のIO/バイト値の確認。
DDT VERS 2.2	
-DO,F /	-----アドレス0~FHをダンプ。
0000 C3 03 DA 83 00 C3 00 BC FF 00 FF 00 FF 00 FF 00	
-^C	↑ IO/バイト。
A>	

Figure-2.2.11 実行前のDDTによるIOバイトのダンプ。

このように, "RDR:", "PUN:" 共にTTY:" になっています。

では, 実習プログラム "7-8.COM" を実行してみましょう。その実行例を次に示します。

A>7-8/ ...実習プログラムの実行。"RDR:"と"PUN:"に"UR1:", "UP1:"をアサインする。

B3 ...実行前のIOバイトの値。

AB ...実行後のIOバイトの値。

A> (注: このプログラムをもう一度実行する場合には, 事前に, リセット・ボタンによるCP/Mの再起動などで, IOバイトを初期値にセットし直して下さい。さもないと同じ値が表示されます。)

Figure-2.2.12 ファンクション: 7, 8 実習プログラムの実行。

このように, 実行前のIOバイトの値は "83H" であり, 実行後は "ABH" に変更されていることが表示されています。

83H = 10000011 Figure-2.2.10のアサイン。

ABH = 10101011 Figure-2.2.13のアサイン。

このように, プログラムの目的通りにIOバイトが変更されましたが, これをSTATコマンドでも確認してみましょう。

A>STAT DEV: /STATコマンドで, 実習プログラム実行後の確認をする。

CON: is UC1:

RDR: is UR1: TTY: →UR1: } と変更されている。

PUN: is UP1: TTY: →UP1: }

LST: is LPT:

A>

Figure-2.2.13 実習プログラムの実行後のSTATコマンドによる結果の確認。

一応, DDTコマンドで, IOバイト付近をダンプしてみましょう。

A>DDT /DDTによる実行後のIOバイト値の確認。

DDT VERS 2.2

-D0, F / 0~FHをダンプ。

0000 C3 03 DA AB 00 C3 00 BC FF 00 FF 00 FF 00 FF 00

-^C

↑
IOバイト。

A>

Figure-2.2.14 実行後のDDTによるIOバイトのダンプ。

目的通りにアサインされていることが確認されました。

このファンクション：7，8により，ユーザー・プログラム中で，自由にフィジカル・デバイスの選択が可能になる訳です。

ファンクション：9の実習

ファンクション：9…文字列のプリントアウト

CALL手順

```

[ MVI    C, 9
  (D, E) ←文字列アドレス
  CALL   0005H ]
    
```

機能

文字列が格納されているメモリの先頭アドレスを (D, E) レジスタにセットしてCALLすると，文字列の最後を示す "\$" 記号 (24H) が来るまで，文字列をコンソールに出力する。

Ctrl-S, Ctrl-P, タブ処理などのコントロールは，ファンクション：2と同様に有効である。

実習プログラム ファンクション：9

プログラムを実行すると，MorningのMか，またはNightのNかをキーインするようメッセージが表示される。

そこで，Mをキーインすると，"GOOD MORNING……"と応答があり，Nをキーインすると"GOOD NIGHT……"と応答する。

Ctrl-Zをキーインすると，プログラムを終了してCP/Mへ戻る。

このようなプログラムを作ってみましょう。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 9.PRN /

```

;-----;
; FUNCTION 9: PRINT STRING ;
;-----;
    
```

0100

ORG 100H

START:			
0100 0E09	MVI	C, 9	"文字列プリント"のシステム・コール。 (DE)レジスタに、プリント・アウトする 文字列の格納バッファのアドレスをセッ トすること。
0102 113A01	LXI	D, MSGINF	
0105 CD0500	CALL	0005H	
0108 0E01	MVI	C, 1	"コンソール入力"のシステム・コール。
010A CD0500	CALL	0005H	
010D FE4D	CPI	'M'	入力が"M"なら"MONGOUT"へジャンプ。
010F CA2401	JZ	MONGOUT	
0112 FE4E	CPI	'N'	入力が"N"なら"NIGTOUT"へジャンプ。
0114 CA2F01	JZ	NIGTOUT	
0117 FE1A	CPI	1AH ; ^Z	入力がCtrl-Zなら本プログラムを終了して CP/Mへ戻る。
0119 CB	RZ		
011A 0E02	MVI	C, 2	入力が上記のいずれでもない場合、"?"を 出力する。"コンソール出力"のシステム・ コール。
011C 1E3F	MVI	E, '?'	
011E CD0500	CALL	0005H	
0121 C30001	JMP	START	繰り返す。
MONGOUT:			
0124 0E09	MVI	C, 9	"文字列プリント"のシステム・コール。 文字列バッファ"MSGMONG"の内容が出力される。
0126 115E01	LXI	D, MSGMONG	
0129 CD0500	CALL	0005H	
012C C30001	JMP	START	
NIGTOUT:			
012F 0E09	MVI	C, 9	同上。 文字列バッファ"MSGNIGT"の内容が出力される。
0131 118501	LXI	D, MSGNIGT	
0134 CD0500	CALL	0005H	
0137 C30001	JMP	START	
013A 0D0A696E70	MSGINF: DB	0DH, 0AH, 'input Morning or Night ----> \$'	"文字列プリント"のシステム・コールにより、それぞれの文字列バッファ の内容が"\$"が来るまでコンソールに出力される。0DHは復帰、0AH は改行のコードである。
015E 0D0A0A474F	MSGMONG: DB	0DH, 0AH, 0AH, 'GOOD MORNING! THIS IS FUNCTION 9.', 0DH, 0AH, '\$'	
0185 0D0A0A474F	MSGNIGT: DB	0DH, 0AH, 0AH, 'GOOD NIGHT! THIS IS FUNCTION 9.', 0DH, 0AH, '\$'	
01AA	END		

文字列バッファ・エリア

Figure-2.2.15 ファンクション：9 実習プログラムのPRN形式のソース・リスト。

このソース・リストからアセンブリ・ソース・ファイルを作り、"9.COM" を生成して、プログラムを実行してみましょう。

ファンクション：9実習プログラムの実行

実行例を次に示します。適当に、MやN、その他の文字などをキーインして試してみてください。最後はCtrl-ZでCP/Mに戻ることができます。

A>9) -----実習プログラムの実行。

input Morning or Night ---->M -----"M"をキーイン。

GOOD MORNING! THIS IS FUNCTION 9. -----プログラムの応答。

input Morning or Night ---->N -----"N"をキーイン。

GOOD NIGHT! THIS IS FUNCTION 9. -----プログラムの応答。

input Morning or Night ---->X? -----"X"をキーイン。"?"が表示された。

input Morning or Night ---->^Z -----Ctrl-Zのキーインにより、本プログラムを、
A> 終了してCP/Mに戻る。

Figure-2.2.16 ファンクション：9 実習プログラムの実行。

ファンクション：10の実習

ファンクション：10…コンソール・バッファへの読み込み

CALL手順

```

[ MVI    C, 10…… (=0AH)
  (D, E) ←バッファ・アドレス
  CALL   0005H
  ↓
  (入力された文字列が、コンソール・バッファに格納されている) ]

```

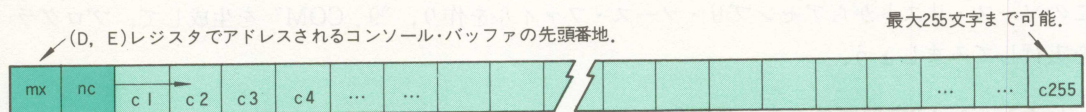
機能

コンソールからの入力を、(D, E)レジスタによってアドレスされるコンソール・バッファの先頭番地＋2番地から、順次、格納して行く。

バッファの入力可能最大文字数は255文字であり、入力に際して、RUB／DEL, Ctrl-C, Ctrl-E, Ctrl-H, Ctrl-J, Ctrl-M, Ctrl-R, Ctrl-Xなどの各種ライン・エディッティング機能が働く。

CALLから戻る条件は、キャリッジ・リターンが入力された時、あるいは“mx”で任意に設定される入力許可文字数（上限は255文字）と、実際に入力された文字数が一致したときである。

コンソール・バッファのフォーマットを次に示す。



mx…ここに、入力許可最大文字数(1～255の間で任意)を、あらかじめセットしておく。

nc…CALLから戻った時に、実際に入力された文字数がセットされる。

c1～c255…入力された文字が格納される。

実習プログラム ファンクション：10

ファンクション：10の機能を確認するには、コンソール・バッファをDDTのロードによって、壊されないアドレス2000H以上の適当な位置に設定し、ファンクション：10を実行した後、DDTでそのコンソール・バッファをダンプすれば、確認することができます。しかし、ここではプログラムを作って、前述のmx, ncの値と、c1以後の入力された文字列とを表示させて、その働きを見ることにしましょう。

コンソール・バッファをアドレス5000Hに設定し、入力許可文字数は40字に制限しておく。

プログラムを実行すると、まず、入力を促すプロンプト “*” が出力され、続けて適当に文字列をキー入力して行く。キャリッジ・リタンで入力を終了すると、コンソール・バッファの内容が、

mx_nc_c1, c2…… (実際に入力された文字列)

と表示され、再びプロンプトに戻る。

プログラムの終了は、当ファンクションのCtrl-C入力を利用し、リブートして終わる。

このようなプログラムを作ってみましょう。

この実習プログラムのPRN形式のソース・リストを示します。

A>TYPE 10.PRN I

```

;-----;
;  FUNCTION 10: READ CONSOLE BUFFER  ;
;-----;

```

0100		ORG	100H	
5000 =	BUFF	EQU	5000H	
0100 3E2B		MVI	A, 40	入力バッファのTOPに値40(=28H)をストア。
0102 320050		STA	BUFF	入力バッファを最大40文字に制限する(255字まで可能)。
	START:			
0105 CD7601		CALL	CRLFOUT	復帰・改行を出力。
010B 0E02		MVI	C, 2	”コンソール出力”のシステム・コール。 ”*”を表示する。
010A 1E2A		MVI	E, '*'	
010C CD0500		CALL	0005H	
010F 0E0A		MVI	C, 10	”コンソール・バッファへの読み込み”のシステム・ コール。バッファへの入力モードとなり、ライン・ エディット、その他のコントロール・キャラ クタが機能する。
0111 110050		LXI	D, BUFF	
0114 CD0500		CALL	0005H	
0117 CD7601		CALL	CRLFOUT	復帰・改行を出力。
011A CD7601		CALL	CRLFOUT	
011D 3A0050		LDA	BUFF	入力バッファの1文字目(入力許可文字数)を 16進表示。
0120 CD5701		CALL	HEXDSPY	
0123 3E20		MVI	A, ' '	スペースを出力。
0125 CD6F01		CALL	CONOUT	
012B 3A0150		LDA	BUFF+1	入力バッファの2文字目(入力された文字数) を16進表示。
012B CD5701		CALL	HEXDSPY	
012E 3E20		MVI	A, ' '	スペースを出力。
0130 CD6F01		CALL	CONOUT	


```

0133 210250      LXI      H, BUFF+2
0136 3A0150      LDA      BUFF+1
0139 FE00        CPI      00
013B CA4E01      JZ       J1
013E 47          MOV      B, A

DSPLY:
013F C5E5        PUSH     B! PUSH H
0141 0E02        MVI      C, 2
0143 5E          MOV      E, M
0144 CD0500      CALL     0005H
0147 E1C1        POP      H! POP B

0149 23          INX      H
014A 05          DCR      B
014B C23F01      JNZ      DSPLY

014E CD7601      J1:      CALL     CRLFOUT
0151 CD7601      CALL     CRLFOUT
0154 C30501      JMP      START

HEXDSPLY:
0157 F5          PUSH     PSW
0158 0F0F0F0F    RRC! RRC! RRC! RRC
015C CD6001      CALL     HOUT1
015F F1          POP      PSW
0160 E60F        ANI      0FH
0162 C630        ADI      30H
0164 FE3A        CPI      3AH
0166 DA6B01      JC       HOUT2
0169 C607        ADI      7
016B CD6F01      HOUT2:   CALL     CONOUT
016E C9          RET

016F 0E02        CONOUT:  MVI      C, 2
0171 5F          MOV      E, A
0172 CD0500      CALL     0005H
0175 C9          RET

CRLFOUT:
0176 3E0D        MVI      A, 0DH
0178 CD6F01      CALL     CONOUT
017B 3E0A        MVI      A, 0AH
017D CD6F01      CALL     CONOUT
0180 C9          RET

0181            END

```

入力された内容を、入力された文字数分
("BUFF+1"の値)表示する。
よって全体の表示は、
(入力バッファ最大文字数)_(入力された文字数)
_(入力された内容)となる。

入力された内容の表示が終わったら、最初から
プログラムの繰り返し。

(A)レジスタの値を16進で表示するサブルーチン。

(A)レジスタのキャラクタをコンソールに出力
するサブルーチン。

復帰・改行をコンソールに出力する
サブルーチン。

A>

Figure-2.2.17 ファンクション：10 実習プログラムのPRN形式のソース・リスト。

このソース・リストからアセンブリ・ソース・ファイルを作り、"10.COM" を生成してプログラムを実行してみましょう。

ファンクション：10 実習プログラムの実行

実行例を次に示します。自由に、文字列を入力したり、ライン・エディティングなどのコントロール・キャラクタを入力したりして実験してみます。

```

A>10J -----実習プログラムの実行.

*ABCDEFGJ -----をキーイン.

2B 07 ABCDEFG ← 入力された内容.
    ↑
    | 入力された文字数. 7文字入力されたことを示している.
    |
    | 入力バッファ許可文字数. 28H=40字(最大255字可能).
    |
*0123456789VWXYZJ -----をキーイン.

2B 0F 0123456789VWXYZ ← 入力された内容.
    ↑
    | 15文字入力されたことを示している.
    |
    | ^Rをキーイン(リタイプさせる). "#はCP/Mの応答.
    |
*abcdefghijklmnop ←
    |
    | ^Rによる表示.
    |
    | 4回RUBOUTをキーイン(4文字削除, エコー・バックされている).
    |
2B 03 abc ← 入力された内容.
    |
    | 結局3文字入字された.

*OPQRSTU#
    |
    | ^Uをキーイン. "#はCP/Mの応答.
    |
J
2B 00
    |
    | すべて削除されて, 何も入力されていない.

*1234567890123456789012345678901234567890 -----
    |
    | 1をキーインしていないことに注目.
    |
    | 任意の文字を40字キーイン. 入力が
    | 設定最大文字数に達すると, 自動的
    | に入力を終了する.
2B 2B 1234567890123456789012345678901234567890
    |
    | 28H=40. バッファがフルになっていることが分かる.

*^C -----1文字目にキーインすると, Ctrl-Cも有効である(2文字目以後は無効).

A> -----リポートしてCP/Mに戻った.

```

Figure-2.2.18 ファンクション：10 実習プログラムの実行例.

なお、このプログラムは、コンソール・バッファを5000Hに設定してありますので、Ctrl-Cでリポートした後(1文字目に入力しないとCtrl-Cは有効でない)、DDTを起動し、5000Hからダンプしてコンソール・バッファを直接確認するのも良いでしょう。

この、ファンクション：10は、ライン・エディティング機能や、各種コントロール・キーによる制御が有効なので、応用プログラム上で文字列の入力に用いると大変便利です。

ファンクション：11の実習

ファンクション：11…コンソール入力のステータスのチェック

CALL手順

MVI C, 11…… (=0BH)

CALL 0005H

↓

(結果はAレジスタに格納されている)

機能

コンソールに入力があったかどうかのチェックを行い、入力があった場合はAレジスタにFFHを持ってCALLから戻る。入力がなかった場合は00を持って戻る。

実習プログラム ファンクション：11

コンソールに、“-”記号を連続して出力させながらコンソール入力のチェックを行い、入力があれば、その文字をコンソールに出力して“-”記号を出力し続ける。

入力が“S”の場合は、次に何らかのキャラクタが入力されるまで“-”記号の出力をストップする。入力がCtrl-Zの場合は、プログラムを終了してCP/Mに戻る。

このようなプログラムを作ってみましょう。

この実習プログラムのPRN形式のソース・リストを次に示します。

```
A>TYPE 11.PRN /
;-----;
;  FUNCTION 11: GET CONSOLE STATUS  ;
;-----;

0100                                ORG      100H

                                START:
0100 0E02                        MVI      C, 2
0102 1E2D                        MVI      E, '-'
0104 CD0500                      CALL     0005H
```

“コンソール出力”のシステム・コール。
“-”記号をコンソールに出力する。

0107 0E0B	MVI	C, 11	"コンソール・ステータスの取り出し"のシステム・コール。入力があった場合はFFH, ない場合は00Hが、(A)レジスタに得られる。
0109 CD0500	CALL	0005H	
010C B7	ORA	A	入力が無い場合は、最初の"START"へジャンプしてループする。
010D CA0001	JZ	START	
0110 0E01	MVI	C, 1	"コンソール入力"のシステム・コール。上記ループで入力があれば、(A)レジスタに入力キャラクタが得られ、同時にコンソールに表示される。
0112 CD0500	CALL	0005H	
0115 FE1A	CPI	1AH ; = ^Z	入力がCtrl-Zなら本プログラムを終了してCP/Mに戻る。
0117 CB	RZ		
0118 FE53	CPI	'S'	入力が"S"以外の場合は、"START"にジャンプしてループする。
011A C20001	JNZ	START	
011D 0E01	MVI	C, 1	入力が"S"の場合、この"コンソール入力"のシステム・コールにより、入力待ちとなる。
011F CD0500	CALL	0005H	
0122 C30001	JMP	START	コンソール入力があれば、"START"に戻りループする。
0125	END		

A>

Figure-2.2.19 ファンクション：11 実習プログラムのPRN形式のソース・リスト

このソース・リストからアセンブリ・ソース・ファイルを作り、"11.COM" を生成して、プログラムを実行してみましょう。

ファンクション：11 実習プログラムの実行

実行例を次に示します。

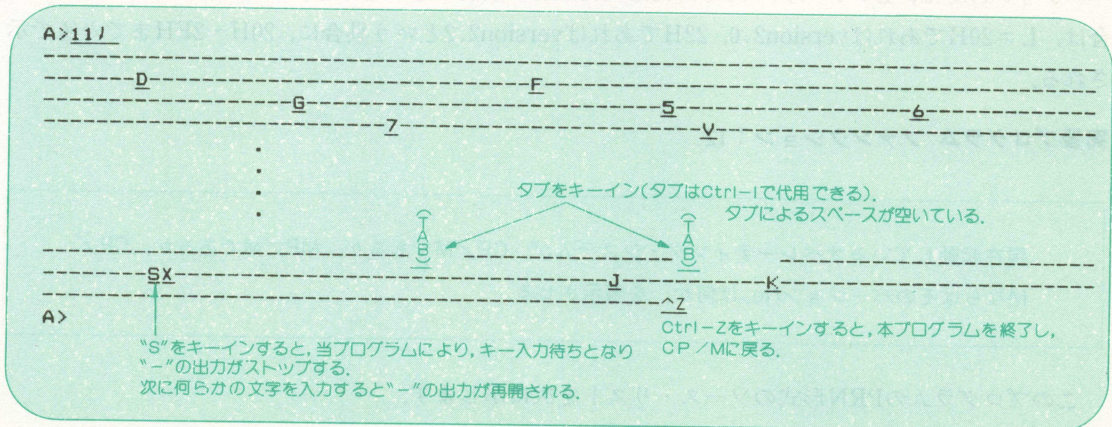


Figure-2.2.20 ファンクション：11 実習プログラムの実行。

プログラムで使われているコンソール出力は、ファンクション：2を使っているので、Ctrl-Sによるフリーズ、Ctrl-Pによるプリンタへの同時出力が有効ですので試して下さい。“S”による“-”表示のストップは、当プログラムによるものです。

このファンクション：11は、何らかのプログラムを実行させながら、キーボードから入力があるかどうかをセンスして、その入力キャラクタにより、それぞれの仕事に分岐させる、というようなプログラムに応用できます。

ファンクション：12の実習

ファンクション：12…オペレーティング・システムのバージョンNo.の取り出し

CALL手順

MVI C, 12 (=0CH) CALL 0005H <div style="text-align: center;">↓</div> (結果はHレジスタおよびLレジスタに格納されている)	
---	--

機能

現在起動しているオペレーティング・システムが、CP/Mであるか、MP/Mであるかの判別を行い、CP/Mであれば、そのバージョンNo.を知らせる。

CALLから戻った時、Hレジスタが00ならばCP/M、01ならばMP/Mであることを示す。

バージョンNo.は、Lレジスタ=00であればversion2.0以前のもを示し、2.0以上のバージョンの場合は、L=20Hであればversion2.0、22Hであればversion2.2という具合に、20H～2FHまでの値で示される。

実習プログラム ファンクション：12

現在起動しているオペレーティング・システムが、CP/Mであるか、MP/Mであるか、CP/MならばそのバージョンNo.は何か、を表示させる。

このプログラムのPRN形式のソース・リストを示します。

A>TYPE 12.PRN)

```

-----
FUNCTION 12: RETURN VERSION NUMBER
-----

```

0100 ORG 100H

START:

```

0100 0E0C MVI C,12
0102 CD0500 CALL 0005H } "バージョンNo. の取り出し"のシステム・コール。

```

```

0105 7C MOV A,H
0106 FE00 CPI 00 } (H)レジスタの結果が00の場合はCP/M、
0108 CA1C01 JZ CPM } "CP/M"へジャンプ。
010B FE01 CPI 01 } (H)レジスタの結果が01の場合はMP/M、
010D CA1601 JZ MPM } "MP/M"へジャンプ。

```

```

0110 117F01 LXI D,MSGANTR
0113 C35001 JMP MSGOUT } どちらでもない場合は、他のシステムなので、
"SYSTEM=ANOTHER"と表示へ。

```

MPM:

```

0116 115601 LXI D,MSGMPM
0119 C35001 JMP MSGOUT } "SYSTEM=MP/M"と表示へ。

```

CPM:

```

011C 116801 LXI D,MSGCPM
011F E5 PUSH H
0120 CD5001 CALL H
0123 E1 POP H } "SYSTEM=CP/M version"と表示へ。
(H)レジスタにはバージョンNo. を持っている
ので保護している。

```

```

0124 7D MOV A,L
0125 FE00 CPI 00 } (L)レジスタが00の場合は、"VER1X"へジャンプ。
0127 CA3801 JZ VER1X } バージョン2.0以前。
012A FE20 CPI 20H } (L)レジスタが20の場合は、"VER20"へジャンプ。
012C CA3E01 JZ VER20 } バージョン2.0。
012F FE22 CPI 22H } (L)レジスタが22の場合は、"VER22"へジャンプ。
0131 CA4401 JZ VER22 } バージョン2.2
0134 D24A01 JNC VER22UP } (L)レジスタが22以上の場合はすべて、
0137 C9 RET } "VER22up"へジャンプ。

```

VER1X:

```

0138 119201 LXI D,MSG1X
013B C35001 JMP MSGOUT } "1.X"と表示へ。

```

VER20:

```

013E 119801 LXI D,MSG20
0141 C35001 JMP MSGOUT } "2.0"と表示へ。

```

VER22:

```

0144 119E01 LXI D,MSG22
0147 C35001 JMP MSGOUT } "2.2"と表示へ。

```

VER22UP:

```

014A 11A401 LXI D,MSG22UP
014D C35001 JMP MSGOUT } "2.2 up"と表示へ。

```

MSGOUT:

```

0150 0E09 MVI C,9
0152 CD0500 CALL 0005H } "文字列のプリント"のシステム・コールのサブ
0155 C9 RET } ルーチン。(DE)レジスタに文字列のアドレス
をセットしてCALLする。

```


0156 0D0A535953 MSGMPM: DB	ODH,0AH,'SYSTEM = MP/M',ODH,0AH,'\$'	文字列エリア。
0168 5359535445 MSGCPM: DB	'SYSTEM = CP/M version \$'	
017F 5354535445 MSGANTR:DB	'STSTEM = ANOTHER',ODH,0AH,'\$'	
0192 312E780D0A MSG1X: DB	'1.x',ODH,0AH,'\$'	
0198 322E300D0A MSG20: DB	'2.0',ODH,0AH,'\$'	
019E 322E320D0A MSG22: DB	'2.2',ODH,0AH,'\$'	
01A4 322E322055 MSG22UP:DB	'2.2 UP',ODH,0AH,'\$'	
01AD	END	

Figure-2.2.21 ファンクション：12 実習プログラムのPRN形式のソース・リスト。

上記ソース・リストからアセンブリ・ソース・ファイルを作りアセンブルし、“12.COM”を生成して実行してみましょう。

ファンクション：12実習プログラムの実行

CP/Mの各種バージョン（一般的には、version 1.4/2.0/2.2）をお持ちの方は、それぞれのバージョンについて実験すると良いでしょう。

まず、それぞれのバージョンのシステムを起動した上で、実習プログラムを実行します。システムがMP/Mの場合は、

SYSTEM = MP/M

と表示しますが、CP/Mの場合は、Figure-2.2.23の実行例のようにバージョンNo.も共に表示されます。

リセット

48K CP/M ver 2.2 -----CP/M起動時のオープニング・メッセージ。

A>12 J -----実習プログラムの実行。

SYSTEM = CP/M version 2.2

A> -----注目。バージョンNo.がレポートされている。

Figure-2.2.22 ファンクション：12 実習プログラムの実行。システムがCP/M version2.2の場合。

リセット

48K CP/M ver 1.4 -----CP/M起動時のオープニング・メッセージ。

A>12 J -----実習プログラムの実行。

SYSTEM = CP/M version 1.x

A> -----注目。プログラムにより、2.0以前は1.xとしてレポートされている。

Figure-2.2.23 ファンクション：12 実習プログラムの実行。システムがCP/M version1.4の場合。

このファンクション：12は、例えば、MP/Mでなければ実行できないアプリケーション・プログラムとか、CP/Mのバージョン2.0以上でなければ実行できないプログラムなどに応用して、現在働いているシステムが、適合するかどうかの判別を行わせる場合に使用します。

ファンクション：13の実習

ファンクション：13…ディスク・システムのリセット

CALL手順

```
[ MVI    C, 13  ..... (=0DH) ]
[ CALL   0005H ]
```

機能

すべてのディスク・システムをイニシャル状態にセットする。つまり、すべてのドライブは“R/W”となり、DMAアドレス（後述）は0080Hのデフォルト値にセットされ、ドライブA：が選択される。ただし、ログイン・ディスクは変化しないので、ファンクション：13を実行した時点のログイン・ディスクがA：以外の場合は、いったんドライブA：が選択された後、元のドライブにログインされる。

実習プログラム ファンクション：13

ファンクション：13で、ディスク・システムをリセットするプログラムを作成する。
プログラムが実行された時、ディスク・システムがリセットされたことを知らせるメッセージを出力させる。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 13.PRN

```

;-----;
;  FUNCTION  13:  RESET DISK SYSTEM  ;
;-----;

0100          ORG      100H
START:
0100 0E0D      MVI      C, 13      ; "ディスク・システムのリセット"の
0102 CD0500    CALL     0005H      ; システム・コール
0105 0E09      MVI      C, 9
0107 110E01    LXI      D, MSGRST  ; "文字列プリント"のシステム・コール、
010A CD0500    CALL     0005H      ; ラベル"MSGRST"の文字列がプリント
                                   ; アウトされる。

```



```

010D C9                RET
010E 0D0A444953  MSGRST: DB      0DH,0AH,'DISK SYSTEM WAS RESET',0DH,0AH,'$'
012B                END
A>

```

Figure-2.2.24 ファンクション：13 実習プログラムのPRN形式のソース・リスト.

上記ソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルし、“13.COM”を生成して実行してみましょう。

ファンクション：13 実習プログラムの実行

一度ログインされたり、何らかのアクセスが行われたディスク・ドライブのディスクを交換した場合、CP/Mは自動的に異なったディスクであることを判別し、そのドライブを書き込み禁止(R/O)にして、ディスクのデータが破壊されるのを防ぎます。

通常このような場合は、Ctrl-Cによるリポートを行って、システム全体のリセットを行ってから、書き込みを伴うプログラムを運用するのですが、このファンクション：13を使えば、リポートを行うことなく、ディスク・システムのリセットを行うことができます。

その実験を行ってみましょう。ドライブB:のディスクを交換して、R/Oとなったことを確認した後、ファンクション：13を実行してみます。その様子を次に示します。

(ドライブA:上に“13.COM”と“STAT.COM”がある。)

A>B: / -----ドライブB:にログイン(B:には適当なディスクをセットしておく)。

B>A:STAT / -----STATコマンド実行。

A: R/W, Space: 197k } B:にもログインしているので、A:,B:共に表示される。R/W表示に注目。

B: R/W, Space: 14k }

(ここで、ドライブB:のディスクを、適当な別のディスクと入れ替える。)

B>DIR PIP.COM / -----入れ替えたディスクのディレクトリを読み込ませるために、例えばDIRコマンドを実行する。

B: PIP COM

B>A:STAT / -----再度STATコマンドを実行。

A: R/W, Space: 197k } B:のR/OがR/Wとなっていることに注目。

B: R/O, Space: 14k } 残りバイト数が異なるディスクであるのに以前のまま(14k)であることに注目。

B>A:13 / -----実習プログラムの実行。

DISK SYSTEM WAS RESET -----実習プログラムによるメッセージ。ファンクション：13が実行された。

B>A:STAT / -----再度STATコマンドを実行。

A: R/W, Space: 197k } B:のR/OがR/Wにリセットされ、入れ替えたディスクの残りバイト数も

B: R/W, Space: 57k } 正しく表示されている。

B>

Figure-2.2.25 ファンクション：13 実習プログラムの実行.

ファンクション：14,17,26の実習

ファンクション：14…ディスク・ドライブの選択

CALL手順

```

[ MVI      C, 14…… (=0EH)
  (E) ← ドライブNo. (0=A:, 1=B:, 2=C:……)
  CALL     0005H ]

```

機能

このファンクション：14が実行された後の各種ファイル操作は、Eレジスタの値によって指定された（ドライブA:=00, B:=01, C:=02, ……）ドライブ上で行われる。つまり、ファンクション：14によって選択されたドライブが、デフォルト・ドライブとなる。新しく選択されたドライブは、当ファンクションにより再選択が行われるか、リブートされるまでは変化しない。

ただし、ファイル操作が行われるファイルのFCB（ファイル・コントロール・ブロック）の最初のバイト（ディスク・ドライブ・コード）が“00”以外の場合は、その値による直接ドライブ選択の方が優先し、ファンクション：14による選択は無視される。

ファンクション：17…最初のファイル・ディレクトリのサーチ

CALL手順

```

[ MVI      C, 17 …… (=11H)
  (D, E) ← FCBアドレス
  CALL     0005H
  ↓
  (結果はAレジスタに格納されているディレクト・コード、およびDMAバッファに格納さ
  れているディレクトリ内容で示される) ]

```

機能

(D, E) レジスタにセットされたFCBで示されるファイルをディレクトリから捜し出し、見つかった場合は、該当ファイルが含まれるディレクトリ・エントリの128バイトをDMAバッファに格納する。その場合、Aレジスタには、0, 1, 2, 3のいずれかの値のディレクトリ・コードがセットされており、その値を32倍（RLCを5回行う）すると、DMAバッファ内の該当ファイルのディレクトリ部の先頭アドレスを知ることができる（DMAアドレスの先頭番地+Aレジスタの値×32=該当ファイルのディレクトリ先頭番地）。

もし、FCBで示されるファイルが見つからなかった場合は、Aレジスタに255 (FFH) がセットされてCALLから戻る。

ファンクション：26…DMAアドレスのセット

CALL手順

```
[ MVI    C, 26  ... (=1AH)
  (D, E) ←DMAアドレス
  CALL   0005H ]
```

機能

DMA (Direct Memory Address) は、ディスクのすべてのリード／ライト操作を行う場合に必要となる1レコード (128バイト) の入出力バッファ・エリアのことであり、デフォルトはアドレス0080Hに設定されている。ディスクへの、どのようなリード／ライトも、必ずこのDMAバッファを通して行われる。

ファンクション：26により、任意のアドレスにDMAバッファを設定することができ、設定されたアドレスは、当ファンクションが再度実行されるか、リブートされるまでは変化しない。

実習プログラム ファンクション：14, 17, 25

ファンクション：14, 17, 26を使って、DIRコマンドの特定ファイル名サーチ(DIR x: filename. ext)に相当する機能を、ユーザー・プログラムで実現する。
また、見つかったディレクトリのDMAバッファ内の位置も確認してみる (DMAアドレスは4000Hにしておく)。

このプログラムのPRN形式のソース・リストを示します。

```
A>TYPE 14-17-26.PRN/

;-----;
;  FUNCTION  14: SELECT DISK                      ;
;              17: SEARCH FOR FIRST                ;
;              26: SET DMA ADDRESS                  ;
;-----;

0100          ORG      100H

START:
0100 0E09      MVI      C, 9
0102 114501    LXI      D, MSGINP
0105 CD0500    CALL     0005H
```

”文字列アライント”のシステム・コール。
ドライブ名入力のメッセージ出力。


```

010B 0E01      MVI      C,1      } "コンソール入力"のシステム・コール。
010A CD0500    CALL     0005H   } キーボードからドライブ名A, B, C, ...を入力する。

010D E60F      ANI      0000*1111B } A=00, B=01, C=02, ...に変換。
010F 3D        DCR      A

0110 0E0E      MVI      C,14     }
0112 5F        MOV      E,A     } "ディスク・ドライブの選択"のシステム・コール。
0113 CD0500    CALL     0005H   } キー入力されたドライブが選択される。

0116 0E1A      MVI      C,26     } "DMAアドレスのセット"のシステム・コール。
011B 110040    LXI      D,4000H  } デフォルトは80Hであるが、DDTでの確認が可
011B CD0500    CALL     0005H   } 能なように4000Hにしておく。

011E 0E11      MVI      C,17     } "最初のディレクトリのサーチ"のシステム・コール。
0120 115C00    LXI      D,005CH  } この例では、FCBアドレスは、デフォルトの5CHに
0123 CD0500    CALL     0005H   } しておく。

0126 FEFF      CPI      255      } 該当ファイルがない場合は、"NAINAI"へ
012B CA3C01    JZ       NAINAI   } ジャンプ。

012B 0707070707 RLC! RLC! RLC! RLC! RLC! } 見つかったファイルのディレクトリの位置を
0130 329040    STA      4090H    } アドレス4090Hに格納する(参考までに)。

0133 0E09      MVI      C,9      }
0135 118101    LXI      D,MSGARI } "ファイルあった"のメッセージ出力。
013B CD0500    CALL     0005H

013B C9        RET

                                NAINAI:
013C 0E09      MVI      C,9      }
013E 116D01    LXI      D,MSGNAI } "ファイルない"のメッセージ出力。
0141 CD0500    CALL     0005H
0144 C9        RET

0145 0D0A494E50 MSGINP: DB      0DH,0AH,'INPUT DRIVE NAME ( A, B, C,...P ) -->*'
016D 0D0A0A4649 MSGNAI: DB      0DH,0AH,0AH,'FILE NAI! NAI! ',0DH,0AH,'*'
0181 0D0A0A4649 MSGARI: DB      0DH,0AH,0AH,'FILE ATTA! ATTA! ',0DH,0AH,'*'

0197          END
A>

```

Figure-2.2.26 ファンクション：14, 17, 26 実習プログラムのPRN形式のソース・リスト。

上記のソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルして、"14-17-26.COM"を生成して実行してみましょう。

ファンクション：14, 17, 26 実習プログラムの実行

このプログラムは、次のコマンド形式で実行されます。

14-17-26 filename . ext J

“filename . ext”には、ディスク上に存在するかどうかを調べようとするファイルのフルネームをキーインします。

このコマンドを実行すると、どのドライブ上のファイルをサーチの対象とするかを問い合わせてきますので、ドライブ名のA, B, C, …をキーインします。

結果は、“あった”とか“ない”とか、メッセージにより表示されます。

実行例を次に示します。これは、ドライブB：上にファイル“PLMX . COM”がある場合のもので

A>14-17-26 PLMX.COM J実習プログラムを実行して、ファイル“PLMX.COM”を捜す。

INPUT DRIVE NAME (A, B, C, ...P) -->AドライブA：上にあるか？

FILE NAI! NAI!A：上には存在しなかった。

A>14-17-26 PLMX.COM J同様にもう1度実行。

INPUT DRIVE NAME (A, B, C, ...P) -->BドライブB：上にはあるか？

FILE ATTA! ATTA!B：上に“PLMX.COM”があった。

A>DDT JDDTを起動。

DDT VERS 2.2

-D4000 JDMAアドレスの4000Hからをダンプしてみる。

4000	00	58	44	49	52	20	20	20	20	C3	4F	4D	00	00	00	0B	.XDIR	.OM....
4010	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
4020	00	50	49	50	20	20	20	20	20	C3	4F	4D	00	00	00	3A	.PIP	.OM...!
4030	03	04	05	06	07	08	09	0A	00	00	00	00	00	00	00	00
4040	00	50	4C	4D	58	20	20	20	20	C3	4F	4D	00	00	00	34	.PLMX	.OM...4
4050	0B	0C	0D	0E	0F	10	11	00	00	00	00	00	00	00	00	00
4060	00	50	4C	4D	4C	45	58	20	20	D0	4C	4D	00	00	00	29	.PLMLEX	.LM...)
4070	12	13	14	15	16	17	00	00	00	00	00	00	00	00	00	00
4080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
4090	40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

-

このように該当ファイルが見つかった場合は、該当ファイルが含まれているディレクトリ・エントリの128/バイト分が、DMA/バッファに格納される。

この値、40Hは、該当ファイルの位置を表す相対アドレスであり、DMAアドレス+40H=4040Hとなり、該当ファイル“PLMX.COM”の位置を指していることが分かる。

(~印の“C3”は、本来“43”=文字のCであるが、このファイルにR/Oアトリビュートをつけてあるため最上位bitが1になり43→C3になっている。)

Figure-2.2.27 ファンクション：14, 17, 26 実習プログラムの実行と、DMAバッファのDDTによるダンプ。

もう1つの実験として“filename.ext”にドライブ名を付け、“x:filename.ext”として実行するとどうなるでしょう。

“x:filename.ext”に付けたドライブx:が優先して、この後ドライブ名入力は、何を入力しても無視されることが確認できます。各自試みて下さい。

ファンクション：15, 20の実習

ファンクション：15…ファイルのオープン

CALL手順

```

MVI    C, 15…… (=0FH)
(D, E) ← FCBアドレス
CALL   0005H
      ↓
(オープンが正常に行われた時 Aレジスタ=00~03, オープンができなかった時 Aレジスタ=255=FFHとなる)

```

機能

(D, E)レジスタで示されるFCBアドレスのファイル名を、該当ディスク上のディレクトリから探し、一致するファイルがあれば、そのディレクトリ・データ (d0~dnフィールド) をメモリ上のFCBにコピーし、今後のリード/ライト操作を可能にする。このことをファイルがオープンされたと言い、正常にオープンされた場合は、Aレジスタには00~03の値(ディレクトリ・コード)がセットされており、もしファイルが見つからずオープンできなかった場合は、Aレジスタには255=FFHがセットされる。FCBのファイル名に、ファイル・マッチ記号“*”および“?”を使用することもできる。ただし、ディレクトリ内の最初にマッチしたファイルがオープンされる。

ファンクション：20…シーケンシャル・リード

CALL手順

```

MVI    C, 20 … (14H)
(D, E) ← FCBアドレス
CALL   0005H
      ↓
(結果は、FCBの“cr”フィールドで示されるレコード (128バイト) が、DMAバッファに読み出される)

```


Aレジスタが00の時は、リードが正常に行われた場合、00でない時は、ファイルが終了した場合。)

機能

(D, E) レジスタでアドレスされるFCBが示す、ファイルのFCB内“cr” フィールドで示される1レコード(128バイト)を読み出して、DMAバッファに格納する。“cr”の値は、自動的に1だけインクリメントされ、次のレコードを示すように更新される。現在のロジカル・エクステント(8インチ標準ディスクでは、16Kバイトが1ロジカル・エクステント)のリードが終了し、まだファイルが続く場合は、自動的に次のロジカル・エクステントがオープンされ、“cr”が00にセットされる。

このファンクション:20を実行するには、事前にFCBアドレスを指定してファイルをオープンし、DMAアドレスをセットしておかなければならない(DMAアドレスをセットしない場合は、デフォルトの80H~FFHがDMAバッファとなる)。さらに、ファイルの最初からリードを行う場合には、“cr”フィールドを00にセットする必要がある。

1レコードのリードが正常に行われた場合、Aレジスタには00が格納されており、エンド・オブ・ファイル(ファイル終了)となった場合は、00以外の値が格納される。これにより、ファイルが終了したことを知ることができる。

リードのみを行うのであれば、オープンしたファイルはクローズする必要はなく、そのままプログラムを終了してよい。

実習プログラム ファンクション:15,20

アスキー・ファイルをタイプアウトするTYPEコマンドに相当するプログラムを作成する。
FCBアドレスはデフォルトの5CH, DMAアドレスは特にセットせず、デフォルトの80Hを使用する。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 15-20.PRN I

```

;-----;
;  FUNCTION   15: OPEN FILE
;              20: READ SEQUENTIAL
;-----;

```

0100 ORG 100H
START:


```

0100 0E0F          MVI    C, 15      "ファイルのオープン"のファンクション・コール。
0102 115C00        LXI    D, 005CH  この例では、FCBアドレスはデフォルトの
0105 CD0500        CALL   0005H      5CH番地にしておく。

0108 FEFF          CPI    255        該当ファイルが存在せず、ファイルのオープ
010A CA3101        JZ     OPNERR     ンができなかった場合は、"OPNERR"へジャンプする。

010D AF           XRA     A          ファイルがオープンされたので、次のシーケ
010E 327C00        STA    5CH+32    ンシャル・リードに備えて、FCBのcrフィールド(5CH+32番地)を0にする。

                                NXTREC:
0111 0E14          MVI    C, 20      "シーケンシャル・リード"のファンクション
0113 115C00        LXI    D, 005CH  ・コール。FCBアドレスは同じく5CHとする。
0114 CD0500        CALL   0005H

0119 FE00          CPI    0          ファイルの最終が来たかどうかのチェック。
011B C0            RNZ             シーケンシャル・リードを終了した場合は、
                                本プログラムを終わり、OP / MIに戻る。

011C 21B000        LXI    H, 00B0H  シーケンシャル・リードのためのDMA/バッ
                                ファは、デフォルトの80H番地である。

                                SCROUT:
011F 0E02          MVI    C, 2
0121 5E            MOV    E, M
0122 E5            PUSH   H
0123 CD0500        CALL   0005H      DMA/バッファに格納されている1レコード
0126 E1            POP    H          (128/バイト)ごとのファイル内容を、次々とコン
                                ソールに出力する。

0127 23            INX     H          128/バイトをコンソールに出力し終えたかのチ
0128 7D            MOV    A, L      ェック。1レコード分をコンソールに出力し
0129 FE00          CPI    0          終えたら、次のレコードをロードするため、
012B C21F01        JNZ    SCROUT   "NXTREC"にジャンプする。
012E C31101        JMP     NXTREC

                                OPNERR:
0131 0E09          MVI    C, 9
0133 113A01        LXI    D, MSGERR  該当ファイルが存在していないなどで、フ
0136 CD0500        CALL   0005H      ァイルがオープンできなかった時は、"ファイル
0139 C9            RET             がない"のメッセージを出力する。

013A 0D0A0A4649    MSGERR:  DB     0DH, 0AH, 0AH, 'FILE GA NAI', 0DH, 0AH, '$'

014B              END

```

A>

Figure-2.2.28 ファンクション：15, 20 実習プログラムのPRN形式のソース・リスト。

上記ソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルし、"15-20.COM" を生成して実行してみましょう。

ファンクション：15, 20実習プログラムの実行

このプログラムを実行するためのコマンド形式は、

15-20_Lx: filename, ext]

であり、TYPEコマンドと同様です。

次に、自分自身のソース・ファイル "15-20. ASM" をタイプアウトする実行例を示します。

```
A>15-20 15-20.ASM / ...ログイン・ディスク上の自分自身のソース・ファイル"15-20.ASM"を
                        タイプアウトする.

;-----;
;  FUNCTION      15: OPEN FILE                      ;
;                20: READ SEQUENTIAL                ;
;-----;

START:  ORG      100H
        MVI      C,15
        LXI      D,005CH
        CALL     0005H

        CPI      255
        JZ       OPNERR
        .
        .
        JNZ      SCRQUT
        JMP      NXTREC

OPNERR: MVI      C,9
        LXI      D,MSGERR
        CALL     0005H
        RET

MSGERR: DB      0DH,0AH,0AH,'FILE GA NAI',0DH,0AH,'$'

        END

A>
```

TYPEコマンドと同様に、アスキー・ファイルがタイプアウトされている。

Figure-2.2.29 ファンクション：15, 20 実習プログラムの実行例.

```
A>15-20 B:15-20.ASM / ...ドライブB：上のファイル"15-20.ASM"をタイプアウトする.
FILE GA NAI ...該当ファイルはドライブB：上にはなかった.

A>
```

Figure-2.2.30 指定したファイルが存在していなかった場合.

タイプアウトしようとするファイルの頭に、任意のドライブ名を指定することができますので、試みて下さい。

注) タイプアウトされたファイルの最後に、スクリーンがクリアされてしまうことなどがありますが、これは、当プログラムの簡素化のために、ファイルの最終のデータ "1AH" を検出せずに、これ以降も最終レコードの128バイトを全部スクリーンに出力してしまうためです。

ファンクション：16,19,21,22の実習

ファンクション：16…ファイルのクローズ

CALL手順

```

MVI    C, 16... (=10H)
(D, E) ←FCBアドレス
CALL   0005H

```



(ファイルのクローズが正常に行われた場合、Aレジスタ=0, 1, 2, 3のいずれかである。クローズしようとするファイルがない場合は、Aレジスタ=255=FFHとなる)

機能

ファイルのオープン (ファンクション：15)、あるいはファイルの作成 (ファンクション：22) のファンクションがすでに実行されており、FCBで示されるファイルに何らかの書き込みが行われた場合、データは、そのつど、1レコードずつディスクに書き込まれて行くが、ディスク上のディレクトリ部に関しては、何も更新されていない。そこで書き込み操作を全部終了した最後の時点で、クローズ・ファンクションを実行し、現在のメモリ上のFCBの内容をディスク上の該当ディレクトリに書き込み、ディレクトリを更新して、新しいファイルとして完成させなければならない。読み出しのみのファイル操作では、オープンしたファイルをクローズする必要はない。

クローズのファンクションが正常に行われた場合、Aレジスタには、ディレクトリ・コードの0, 1, 2, 3のいずれかの値が格納されている。クローズするファイルが存在せず、クローズできなかった場合は、AレジスタはFFHが格納されている。

ファンクション：19…ファイルのデリート

CALL手順

```

MVI    C, 19..... (=13H)
(D, E) ←FCBアドレス
CALL   0005H

```



(該当ファイルの削除が行われた場合は、Aレジスタ=0, 1, 2, 3いずれかの値が格納されており、該当ファイルがなかった場合は、Aレジスタ=255=FFHとなる)

機能

(D, E) レジスタでアドレスされるFCBが示すファイルをディスク上から削除する。ファイル名には

ファイル・マッチ記号 “?” や “*” を使用することができる。また、ドライブ・コード “dr” にドライブNo. を指定することにより、任意のドライブ上のファイルを削除することができる。

該当ファイルが削除された場合は、Aレジスタ=0, 1, 2, 3の、いずれかのディレクトリ・コードが格納されている。該当ファイルが存在せず、削除が行われなかった場合は、Aレジスタ=255=FFHが格納されている。

ファンクション：21…シーケンシャル・ライト

CALL手順

```
MVI    C, 21…… (=15H)
```

```
(D, E) ←FCBアドレス
```

```
CALL    0005H
```



(書き込みが正常に行われた場合、Aレジスタ=00、ディスクがフルで書き込みができなかった場合は、Aレジスタは、00以外の値が格納されている)

機能

ファンクション：20のシーケンシャル・リードと、逆の動作を行う。

(D, E)レジスタでアドレスされるFCBが示すファイルのレコード(“cr”フィールドが示す)に、DMAバッファの128バイトが書き込まれる。“cr”の値は自動的に1だけインクリメントされ、次のレコードの書き込みに備えられる。

現在のロジカル・エクステント(1ロジカル・エクステントは、8インチ標準ディスクの場合、16Kバイト)がフルになった場合(“cr”がオーバ・フローした場合)、自動的に次のロジカル・エクステントがオープンされ、“cr”の値が00にリセットされる。

当ファンクション：21を実行するには、事前に“ファイルのオープン”あるいは“ファイルの作成”ファンクションを実行しておかなければならない。

書き込みが正常に行われた場合、Aレジスタには00が格納されており、ディスクがフルになり、書き込みができなかった場合は、Aレジスタには00以外の値が格納されている。

ファンクション：22…ファイルの作成

CALL手順

```
MVI    C, 22…… (=16H)
```

```
(D, E) ←FCBアドレス
```

```
CALL    0005H
```



(ファイルの作成が行われた場合は、Aレジスタ= 0, 1, 2, 3のいずれか、作成ができなかった場合、Aレジスタ=255=FFHが格納されている)

機能

(D, E) レジスタでアドレスされるFCB上のファイル名を持つファイル (内容はまだ空) を作成する。つまり、新ファイルのディレクトリをディスクに登録する。

作成しようとする新ファイル名は、同一ディスク上にあるファイルと同じファイル名を使ってはいけない。ファイル名が同じものが2つできてしまい、おかしいことになってしまう。このようなことを避けるため、PIPやSAVEコマンドで経験されているように、通常は、すでに存在するファイルと同一名のファイルを、事前にファンクション: 19で削除してから、“ファイルの作成”を実行する。あるいは、EDで行われているようなファイル名のエクステンションを“. \$\$\$”や“. BAK”などに変更するのもよい。

当ファンクションを実行して、ファイルの作成ができた場合は、Aレジスタには、0, 1, 2, 3いずれかが、また、ファイルの数が制限いっぱいファイルが作成できなかった場合は、255=FFHがディレクトリ・コードとして格納される。

“ファイルの作成”によりファイルができると、そのファイルはすでにオープンされた状態にあり、自由にリード/ライト可能である。“ファイルのオープン”のファンクションを再度実行する必要はない。

実習プログラム ファンクション: 16, 19, 21, 22, (20)

PIPコマンドに相当するような、ファイル・コピーのプログラムを作成する。

PIPは一度に16Kバイト分ディスクから読み出しバッファリングするが、当プログラムでは1レコード (128バイト) ずつの“読み出し→書き込み”を行いながらコピーを進める。

このプログラムのPRN形式のソース・リストを示します。

A>TYPE 19-21-22.PRN /

```

;-----;
;  FUNCTION  16: CLOSE FILE      ;
;            19: DELETE FILE    ;
;            21: WRITE SEQUENTIAL;
;            22: MAKE FILE      ;
;-----;

```

```

0100      ORG      100H
005C =    FCB%D   EQU      005CH ...デスティネーション側のFCBアドレスは、デフォルトの5CHに設定、
01FF =    FCB%S   EQU      FCB$S ...ソース側のFCBアドレスは、当プログラムの最後部に設定、

```


START:			
0100 0E10	MVI	C, 10H	アドレス5CHからのFCB内のセカンド・ファイル名(アドレス8CHから格納されている)を、ソース側のFCBアドレスに転送。
0102 216C00	LXI	H, FCB*D+10H	
0105 11FF01	LXI	D, FCB*S	
0108 7E	FCBS*MOV: MOV	A, M	
0109 12	STAX	D	
010A 23	INX	H	
010B 13	INX	D	
010C 0D	DCR	C	
010D C20B01	JNZ	FCBS*MOV	
0110 0E0F	MVI	C, 15	ソース側のFCBで示されるファイルをオープンする。オープンしようとするファイルがない場合は、“OPNERR”へジャンプ。
0112 11FF01	LXI	D, FCB*S	
0115 CD0500	CALL	0005H	
0118 FEFF	CPI	255	
011A CA5601	JZ	OPNERR	
011D 0E13	MVI	C, 19	デスティネーション側(新しく生成する側)のFCBで示されるファイルと同一名のファイルが存在していれば、そのファイルを削除する。
011F 115C00	LXI	D, FCB*D	
0122 CD0500	CALL	0005H	
0125 0E16	MVI	C, 22	デスティネーション側のFCBで示されるファイルを新たに作成する(と同時に、オープン状態になる)。ファイル数が制限いっぱいの場合は、“MAKERR”へジャンプ。
0127 115C00	LXI	D, FCB*D	
012A CD0500	CALL	0005H	
012D FEFF	CPI	255	
012F CA5C01	JZ	MAKERR	
0132 AF	XRA	A	両方のFCBのレコード・カウンタ(“cr”フィールド)を00にセットする。
0133 321F02	STA	FCB*S+32	
0136 327C00	STA	FCB*D+32	
NXTREC:			
0139 0E14	MVI	C, 20	ソース・ファイルを1レコード読み出す。DMAアドレスはデフォルト値の80Hであるので読み出されたデータは80H~FFHに格納される。ファイル・エンドになった場合は、“CPYEND”へジャンプ。
013B 11FF01	LXI	D, FCB*S	
013E CD0500	CALL	0005H	
0141 FE00	CPI	0	
0143 C26B01	JNZ	CPYEND	
0146 0E15	MVI	C, 21	DMA/バッファの1レコード分をディスクに書き込む。ディスクがフルになって空きスペースがない場合は、“WRTERR”へジャンプ。
0148 115C00	LXI	D, FCB*D	
014B CD0500	CALL	0005H	
014E FE00	CPI	0	
0150 C26201	JNZ	WRTERR	
0153 C33901	JMP	NXTREC	次のレコードの読み出しへループ。
OPNERR:			
0156 11B401	LXI	D, MSGSER	ソース・ファイルがない場合のエラー・メッセージ出力。
0159 C37E01	JMP	MSGOUT	
MAKERR:			
015C 119C01	LXI	D, MSGMER	ファイルの数が制限いっぱいの場合のエラー・メッセージ出力。
015F C37E01	JMP	MSGOUT	
WRTERR:			
0162 11BA01	LXI	D, MSGWER	ディスクがフルになり、空きエリアがない場合のエラー・メッセージ。
0165 C37E01	JMP	MSGOUT	
CPYEND:			
0168 0E10	MVI	C, 16	コピーの書き込みが全部終了したので、デスティネーション・ファイルをクローズする。クローズできない場合は、“CLSERR”へジャンプ。
016A 115C00	LXI	D, FCB*D	
016D CD0500	CALL	0005H	
0170 FEFF	CPI	255	
0172 CA7B01	JZ	CLSERR	
0175 11E901	LXI	D, MSGEND	コピー作業終了のメッセージ出力。
017B C37E01	JMP	MSGOUT	
CLSERR:			
017B 11D101	LXI	D, MSGCER	クローズ・エラーのメッセージ出力。


```

MSGOUT:
017E OE09      MVI    C,9
0180 CD0500    CALL   0005H
0183 C9        RET

0184 ODOA0A534F MSGSER: DB    0DH,0AH,0AH,'SOURCE FILE GA NAI',0DH,0AH,'*'
019C ODOA0A4649 MSGMER: DB    0DH,0AH,0AH,'FILE DIRECTORY GA MANPAI',0DH,0AH,'*'
01BA ODOA0A4449 MSGWER: DB    0DH,0AH,0AH,'DISK SPACE GA NAI',0DH,0AH,'*'
01D1 ODOA0A4649 MSGCER: DB    0DH,0AH,0AH,'FILE CLOSE DEKINAI',0DH,0AH,'*'
01E9 ODOA0A434F MSGEND: DB    0DH,0AH,0AH,'COPY DEKIMASHITA',0DH,0AH,'*'

01FF          FCB$S:  DS      OFH
020E          END

A>

```

メッセージ出力のサブルーチン。

Figure-2.2.31 ファンクション：16, 19, 21, 22 実習プログラムのPRN形式のソース・リスト。

上記ソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルし、“19-21-22.COM”を生成して実行してみましょう。

ファンクション：16, 19, 21, 22実習プログラムの実行

当プログラムを実行するためのコマンドの書式を次に示します。

19-21-22 x : filename1 . ext x ' : filename2 . ext]
 (新しく生成される側) (ソース側)

このコマンドを実行することにより、ドライブ x : 上のファイル “filename2 . ext” が、ドライブ x : 上にファイル “filename1 . ext” としてコピーされます。

実行例として、ドライブ A : 上のファイル “DUMP . ASM” を、ドライブ B : 上に、ファイル名を “DUMPCPY . ASM” としてコピーした例を示します。

A>19-21-22 B:DUMPCPY.ASM DUMP.ASM] …filename 1 と filename 2 の順序に注意。

COPY DEKIMASHITA …コピーが完了したメッセージ。

A>DIR B:DUMPCPY.ASM] …ドライブ B : 上に “DUMPCPY.ASM” としてコピーされたファイルの確認。
 B: DUMPCPY ASM …ファイルが作られていた。

A>TYPE B:DUMPCPY.ASM] ……新しく作られたファイルの内容の確認。


```

; FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
; COPYRIGHT (C) 1975, 1976, 1977, 1978
; DIGITAL RESEARCH
; BOX 579, PACIFIC GROVE
; CALIFORNIA, 93950
;
ORG      100H
BDOS     EQU      0005H      ;DOS ENTRY POINT
CONS     EQU      1         ;READ CONSOLE
TYPEF    EQU      2         ;TYPE FUNCTION
PRINTF   EQU      9         ;BUFFER PRINT ENTRY
BRKF     EQU      11        ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
OPENF    EQU      15        ;FILE OPEN
READF    EQU      20        ;READ FUNCTION
.
.
.
.
.
OPNMSG:  DB        CR,LF,'NO INPUT FILE PRESENT ON DISK*'
;
; VARIABLE AREA
IBP:     DS        2         ;INPUT BUFFER POINTER
OLDSP:   DS        2         ;ENTRY SP VALUE FROM CCP
;
; STACK AREA
DS       64             ;RESERVE 32 LEVEL STACK
STKTOP:
;
END

```

Figure-2.2.32 ファンクション：16, 19, 21, 22 実習プログラムの実行とその確認.

実行するには、ソース側とオブジェクト側（新しく生成される側）のファイル名を逆に指定しないように注意して下さい。1レコード(128バイト)ずつリード／ライトが行われますので、ドライブがせわしなく動作しますがあしからず。

また、オブジェクト側に、すでに存在しているファイルと同一名を指定して実行した場合、元あったファイルが削除されて、新しくコピーされたものになっていることも確認して下さい。

ファンクション：18の実習

ファンクション：18…次のファイル・ディレクトリのサーチ

CALL手順

```

MVI    C, 18…… (=12H)
CALL   0005H

```



(結果はAレジスタに格納されているディレクトリ・コード、およびDMAバッファに格納されているディレクトリの内容で示される)

機能

前回のサーチで一致したファイルから後の部分のディレクトリを対象に、ファイルを探す。その他のことは、ファンクション：17「最初のファイル・ディレクトリのサーチ」と全く同一である。

実習プログラム ファンクション：18, (17)

DIRコマンドに相当するプログラムを作成する。ファンクション：17の「最初のサーチ」を作って、最初に一致したファイル名をタイプアウトし、その後はファンクション：18の「次のサーチ」を使って、一致したファイル名を次々とタイプアウトする。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 17-18.PRN J

```

;-----;
;  FUNCTION  17: SEARCH FOR FIRST          ;
;           18: SEARCH FOR NEXT            ;
;-----;

```

```

0100          ORG      100H
0100 0E11  START:    MVI      C, 17
0102 115C00      LXI      D, 005CH
0105 CD0500      CALL     0005H
                                } 最初のファイルのサーチのシステム・コール。
                                }  FCBアドレスはデフォルトの5CHとする。

010B FEFF          CPI      255
010A CA4301      JZ       NOFILE
                                } ファイルが見つからない場合は、"NOFILE"
                                } ヘジャンプ。

```



```

      NXTFILE:
010D 0707070707      RLC! RLC! RLC! RLC! RLC
0112 5F              MOV     E, A
0113 1600             MVI     D, 0
0115 218000           LXI     H, 0080H
0118 19              DAD     D
0119 23              INX     H      .....ドライブNo. ("dr"フィールド)はここでは省略.

011A 060B            MVI     B, 11 .....ファイル名"xxxxxxxxxx(.)xxxx"で11文字.

      OUTLP:
011C C5E5            PUSH B! PUSH H
011E 0E02            MVI     C, 2
0120 5E              MOV     E, M
0121 CD0500          CALL    0005H
0124 E1C1            POP H! POP B

0126 23              INX     H
0127 05              DCR     B
0128 C21C01          JNZ     OUTLP

012B 0E09            MVI     C, 9
012D 117301          LXI     D, MSGCL
0130 CD0500          CALL    0005H

0133 0E12            MVI     C, 18
0135 CD0500          CALL    0005H

013B FEFF            CPI     255
013A CA4901          JZ      ENDFILE
013D C3D0D1          JMP     NXTFILE

      NOFILE:
0140 0E09            MVI     C, 9
0142 115201          LXI     D, MSGNOF
0145 CD0500          CALL    0005H
0148 C9              RET

      ENDFILE:
0149 0E09            MVI     C, 9
014B 116001          LXI     D, MSGED
014E CD0500          CALL    0005H
0151 C9              RET

0152 0D0A0A4649      MSGNOF: DB      0DH, 0AH, 0AH, 'FILE NAI', 0DH, 0AH, '*'
0160 0D0A0A4649      MSGED:  DB      0DH, 0AH, 0AH, 'FILE OWARI', 0DH, 0AH, '*'
0170 0D0A24          MSGCL:  DB      0DH, 0AH, '*'

```

見つかったファイル・ディレクトリの, DMA
バッファ内(デフォルトの80H~FFH)のアド
レスを求める.

"コンソールへの出力"ファンクションを使用.

11文字出力したかどうかの判断ルーチン.
11文字分ループする.

復帰・改行を出力.

"次のファイルのサーチ"のシステム・コール.
FCBアドレスは同じくデフォルトの5CH.

一致するファイルがさらにあれば"NXTFILE"
へジャンプして, ファイル名をタイプアウト.
なければ"ENDFILE"へジャンプ.

"ファイルがない"のメッセージ出力ルーチン.

"ファイル終了"のメッセージを出力して, 当
プログラムを終了し, CP/Mに戻る.

Figure-2.2.33 ファンクション: 18, 17 実習プログラムのPRN形式のソース・リスト.

上記ソース・リストからアセンブリ・ソース・ファイルを作り, アセンブルして, "17-18.COM"を生成し実行してみましょう.

ファンクション：18, 17 実習プログラムの実行

当プログラムは、次のコマンド形式で実行します。

- 1) 17-18 □ x : filename . ext] ……ドライブ x : 上の特定のファイル名をタイプアウト。
- 2) 17-18 □ x : filemach] ……ドライブ x : 上のファイル・マッチするファイル名をすべてタイプアウト。

次に実行例を示します。

```
A>17-18 PIP.COM / ...ログイン・ディスク上のファイル"PI P.COM"を捜す。
PIP      COM .....ファイルは存在していた。
FILE OWARI
A>
```

Figure-2.2.34 ファンクション：18, 17 実習プログラムの実行。

```
A>17-18 B:*.COM / ...ファイル・マッチ記号を使って、ドライブB:上のエクステンションが
"COM"であるファイルを捜す。
COBOL    COM
RUNA     COM
CONFIG   COM
FILEMARKCOM } これだけあった。
FILE OWARI
A>
```

Figure-2.2.35 プライマリ・ネームにファイル・マッチ記号を使用した実行例。

```
A>17-18 B:*. * / ...ファイル・マッチ記号を使って、ドライブB:上のすべてのファイルを捜す。
COBOL    COM
COBOL    IO1
COBOL    IO2
COBOL    IO3
COBOL    IO4
RUNA     COM
CONFIG   COM
PI       CBL
STOCK1   CBL
STOCK2   CBL
CALL     ASM
CALL     PRN
CALL     HEX
FILEMARKCOM } すべてのファイルがタイプアウトされた。
```



```
FILE OWARI
A>
```

Figure-2.2.36 ファイル・マッチ記号だけを使用した実行例

このように、タイプアウトされたファイル名の頭にドライブ名は付きませんが、DIRコマンドと同様な働きをします (CP/M version1.4のDIRは、上記実行例のように縦一列にファイル名が並んだ)。ファイル・マッチ記号の“?”を使った例も実験してみてください。

ファンクション：23の実習

ファンクション：23…ファイル名の変更

CALL手順

[MVI C, 23…… (17H) (D, E) ←CBアドレス CALL 0005H <div style="text-align: center; margin: 5px 0;">↓</div> (Aレジスタ= 0, 1, 2, 3のいずれか, ……変更が正常に行われた場合. Aレジスタ=255=FFH, ……目的のファイルが存在しなかった場合)]
---	---	---

機能

(D, E) レジスタでアドレスされるFCB内の1stファイル名 (dr~t3フィールド) を, 2ndファイル名 (d₀~d₁₅フィールド) に変更する。

ディスク・ドライブの選択は, 旧ファイル名となる1stファイル名の頭に付けられるドライブ・コード “dr” によって行われ, 新ファイル名のドライブ・コード “d₀” は, 値が入っていても無視される。ファンクション終了時, ファイル名の変更が正常に行われた場合は, Aレジスタには 0, 1, 2, 3 いずれかのディレクトリ・コードが格納され, 変更しようとする目的のファイル (1stファイル名) が存在しなかった場合は, 255=FFHが格納される。

実習プログラム ファンクション：23

RENコマンドに相当するプログラムを作成する。

このプログラムのPRN形式のソース・リストを示します。

A>TYPE 23.PRN /

```

;-----;
;  FUNCTION  23:  RENAME FILE  ;
;-----;

0100          ORG      100H
START:
0100 0E17      MVI      C,23
0102 115C00     LXI      D,005CH
0105 CD0500     CALL     0005H    ;"ファイル名の変更"のシステム・コール。
                                ;FCBアドレスは、デフォルトの5CHとする。

0108 3C         INR      A
0109 CA1501     JZ       ERR      ;目的のファイルがなかった場合は、"ERR"へ
                                ;ジャンプ。

010C 0E09      MVI      C,9
010E 111E01     LXI      D,MSGOK
0111 CD0500     CALL     0005H    ;ファイル名の変更が終了したOKメッセージを
                                ;出力し、CP/Mへ戻る。
0114 C9        RET

ERR:
0115 0E09      MVI      C,9
0117 112501     LXI      D,MSGERR
011A CD0500     CALL     0005H    ;目的のファイルがなかったで、エラー・メッ
                                ;セージを出力し、CP/Mへ戻る。
011D C9        RET

011E 0D0A4F4B0D MSGOK:  DB      0DH,0AH,'OK',0DH,0AH,'*'
0125 0D0A455252 MSGERR: DB      0DH,0AH,'ERROR',0DH,0AH,'*'

012F          END

A>

```

Figure-2.2.37 ファンクション：23 実習プログラムのPRN形式のソース・リスト。

上記ソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルし、"23.COM" を生成して実行してみましょう。

ファンクション：23 実習プログラムの実行

当プログラムは、次のコマンド形式で実行します。

23 x : oldfilename . ext newfilename . ext / ドライブ x : 上の "旧ファイル名" を、
"新ファイル名" に変更する。

REN コマンドとは、新・旧ファイルの順序が異なりますので注意して下さい。実行例を示します。

A>DIR /実習プログラム実行前の全ファイルを確認しておく。

```
A: SC          COM : SC          OVL : INSTALL  COM : INSTALL  DAT
A: INSTALL    OVL : BALANCE    CAL : BARRIER CAL : BRKEVN  CAL
A: INSTALL    SUB : 23          COM
A>
```

実習プログラム。

Figure-2.2.38 実行前のオリジナル・ディスクットの全ファイルをタイプアウトして確認しておく

A>23 SC.COM SUPERCAL.COM /“SC.COM”⇒“SUPERCAL.COM”にリネームする。

OK正常に変更が行われたメッセージ。

A>

Figure-2.2.39 ファンクション23 実習プログラムの実行例①。

A>DIR /結果の確認。

```
A: SUPERCAL COM : SC          OVL : INSTALL  COM : INSTALL  DAT
A: INSTALL    OVL : BALANCE    CAL : BARRIER CAL : BRKEVN  CAL
A: INSTALL    SUB : 23          COM
A>
```

リネームされている。

Figure-2.2.40 実行例①の結果の確認。

A>B: /ログイン・ディスクをB:にチェンジ。

B>A:23 A:SC.OVL C:SUPERCAL.OVL /ドライブA:上の“SC.OVL”を、ドライブ名“C:”を付けた
“SUPERCAL.OVL”にリネームする。
(2ndファイルのドライブ名は無視されることの確認のため)

OK正常に変更が行われた
メッセージ。

B>

Figure-2.2.41 ファンクション:23 実習プログラムの実行例②。

B>DIR A: /結果の確認。

```
A: SUPERCAL COM : SUPERCAL OVL : INSTALL  COM : INSTALL  DAT
A: INSTALL    OVL : BALANCE CAL : BARRIER CAL : BRKEVN  CAL
A: INSTALL    SUB : 23      COM
B>
```

リネームされている。

このように2ndファイルのドライブ名は無視される。

Figure-2.2.42 実行例②の結果の確認。

ファンクション：24…ログイン・ベクトルの取り出し

ファンクション：24…ログイン・ベクトルの取り出し

CALL手順

```

MVI    C, 24…… (=18H)
CALL   0005H
      ↓
(H.Lレジスタのログイン・ベクトルが格納されている)

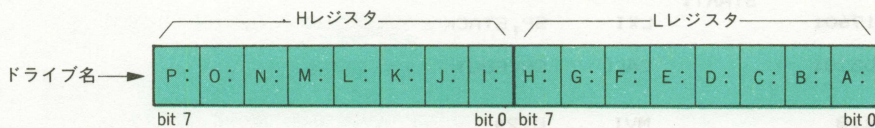
```

機能

現在、オンラインとなっているディスク・ドライブのベクトルを報告する。“オンラインとなっている”とは、CP/Mの起動、あるいはリブート以後にログインされたドライブ、もしくは自動ディスク・セレクト (FCBの “dr” フィールドによる) されたことのあるドライブのことで、これらのドライブは、それぞれのディレクトリ情報がCP/Mに登録済である。

分かりやすい例では、STATコマンドを“STAT J”と実行した場合に結果が表示されるドライブが、オンラインとなっているドライブである。

取り出されたベクトルは、(H, L) レジスタに次のように格納されている。



それぞれのビットが { 0:オフライン } のドライブである。
 { 1:オンライン }

ファンクション：25…ログイン・ディスクNo.の取り出し

CALL手順

```

MVI    C, 25…… (=19H)
CALL   0005H
      ↓
(Aレジスタにログイン・ディスクNoが格納されている)

```


機能

現在選択されているディスク（ログイン・ディスクとか、カレント・ディスクとか呼ぶ）のNo.を報告する。

Aレジスタの値が0の場合ドライブA：，1の場合B：，……15の場合：Pとなる。

実習プログラム ファンクション：25, 26

現在のログイン・ベクトルとログイン・ディスクNo.を取り出して，ログイン・ベクトルが格納されている（H，L）レジスタのビット状態を表示し，ログイン・ディスクNo.が格納されているAレジスタの値を表示する。

このプログラムPRN形式のソース・リストを次に示します。

A>TYPE 24-25.PRN I

```

;-----;
;  FUNCTION  24: RETURN LOGIN VECTOR      ;
;           25: RETURN CURRENT DISK      ;
;-----;

0100          ORG      100H
0100 319601    START: LXI      SP,STACK  ;当プログラムは、スタックが深くなるので、
                                ;新たに設定する。
0103 CD5201    CALL     CRLFOUT  ;復帰・改行をしておく。

0106 0E18      MVI      C,24
0108 CD0500    CALL     0005H  ;"ログイン・ベクトルの取り出し"のシステム・
                                ;コール。
                                ;H,Lレジスタにベクトルが得られる。

010B 7C        MOV      A,H
010C CD2C01    CALL     BITOUT  ;Hレジスタのビット・パターンを出力。
010F 1E24      MVI      E,'$'  ;HレジスタとLレジスタを区別するために
0111 CD4601    CALL     CONOUT  ;"$"記号を出力。
0114 7D        MOV      A,L
0115 CD2C01    CALL     BITOUT  ;Lレジスタのビット・パターンを出力。

0118 CD5201    CALL     CRLFOUT ;復帰・改行を2度出力。
011B CD5201    CALL     CRLFOUT

011E 0E19      MVI      C,25
0120 CD0500    CALL     0005H  ;"ログイン・ディスクNo.の取り出し"のシステム・
                                ;コール。ドライブNo.がAレジスタに得ら
                                ;れる。

0123 CD5D01    CALL     HEXDSPLY ;Aレジスタの値を16進表示。
0126 CD5201    CALL     CRLFOUT  ;復帰・改行を出力。

0129 C30000    JMP      0000H  ;リポートしてCP/Mに戻る。

```



```

012C 060B      BITOUT:      MVI      B, 0
012E 07        LOOP:      RLC
012F DA3C01    JC          OUT1
0132 1E30      MVI      E, '0'
0134 CD4601    CALL     CONOUT
0137 05        DCR      B
0138 C22E01    JNZ      LOOP
013B C9        RET

013C 1E31      OUT1:      MVI      E, '1'
013E CD4601    CALL     CONOUT
0141 05        DCR      B
0142 C22E01    JNZ      LOOP
0145 C9        RET

0146 F5C5E5    CONOUT:     PUSH PSW! PUSH B! PUSH H
0149 0E02      MVI      C, 2
014B CD0500    CALL     0005H
014E E1C1F1    POP H! POP B! POP PSW
0151 C9        RET

0152 1E0D      CRLFOUT:   MVI      E, 0DH
0154 CD4601    CALL     CONOUT
0157 1E0A      MVI      E, 0AH
0159 CD4601    CALL     CONOUT
015C C9        RET

015D F5        HEXDSPLY:  PUSH     PSW
015E 0F0F0F0F  RRC! RRC! RRC! RRC
0162 CD6601    CALL     HOUT1
0165 F1        POP      PSW
0166 E60F      HOUT1:    ANI      0FH
0168 C630      ADI      30H
016A FE3A      CPI      3AH
016C DA7101    JC          HOUT2
016F C607      ADI      7
0171 5F        HOUT2:    MOV      E, A
0172 CD4601    CALL     CONOUT
0175 C9        RET

0176           DS      32
0196 =         STACK   EQU     $
0196           END

A>

```

レジスタの値を、0-1のビット・パターンで表示するサブルーチン。
ビット7が左側、ビット0が右側として、1回に1ビットずつ順に出力される。

Eレジスタが持つアスキー・コードを、コンソールに出力するサブルーチン。

復帰・改行を出力するサブルーチン。

Aレジスタの値を16進でコンソールに出力するサブルーチン。
例えば値が7FHであれば、コンソールに“7F”と出力する。

スタック・エリア

Figure-2.2.43 ファンクション：24, 25 実習プログラムのPRN形式のソース・リスト。

上記ソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルし、“24-25.COM”を生成して実行してみましょう。

ファンクション：24, 25 実習プログラムの実行

当プログラムを実行するコマンド形式は1種類であり、次に示します。

24-25 / ……ただこれだけ。

次に実行例を示します。他のドライブへのアクセスや、ログイン・ディスクの変更に注目して下さい。

CP / M起動

A>24-25 / ……CP / Mが起動した直後の実行。

00000000*00000001 ……ドライブA：のみオンライン。

00 ……ログイン・ディスク(カレント・ディスク)はA：。

A>DIR B: / ……ドライブB：をアクセスしてみる(例えばDIRなどで)。

B: SC	COM : SC	OVL : INSTALL	COM : INSTALL	DAT
B: INSTALL	OVL : BALANCE	CAL : BARRIER	CAL : BRKEVN	CAL
B: INSTALL	SUB			

A>24-25 / ……再度実行。

00000000*00000011 ……ドライブB：もオンラインとなり、現在 A：とB：がオンラインである。

00 ログイン・ドライブはA：

A>B: / ……ドライブをチェンジして、B：にログインした。

B>A: 24-25 / ……ドライブA：上の当プログラムを実行。

00000000*00000011 ……A：, B：共にオンライン。

01 ……ログイン・ドライブはB：

B>

Figure-2.2.44 ファンクション24, 25 実習プログラムの実行

当プログラムの終了は、リブートで行っています。リブートが行われると、システム全体がイニシャル状態になり、ログイン・ベクトル、ログイン・ドライブも、いったんイニシャライズされます。

ファンクション：27の実習

ファンクション：27…アロケーション・アドレスの取り出し

CALL手順

```
MVI    C, 27…… (=IBH)
```

```
CALL   0005H
```



(H, Lレジスタに、アロケーション・ベクトルのベース・アドレスが格納されている)

機能

現在ログインされているディスク・ドライブのアロケーション・ベクトルのベース・アドレス（先頭アドレス）を報告する。

BIOS内のアロケーション・ベクトル（1.1.2～3章参照）には、それぞれのドライブのディスクのメモリ使用状態が記録されており、このベクトルを調べることで、ディスクのどの部分が使われているか、残りメモリ容量は何バイトか、などを知ることができる。

実習プログラム ファンクション：27

ログイン・ディスクのアロケーション・ベクトルのメモリ内容をダンプするプログラムを作成する。

アロケーション・ベクトルのバッファ・エリアのサイズは、8インチ標準ディスクの場合は31バイトである（ドライブ容量=243Kバイト、アロケーション・ベクトルの1バイトで8Kバイトを表すので、 $243/8=30.375$ となり、31バイトで243Kバイトのマップを表すことが可能である。1.1.2～3章参照）。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 27.PRN J

```

FUNCTION 27: GET ALLOCATION ADDRESS

```

0100

ORG

100H


```

START:
0100 0E1B      MVI    C, 27      "アロケーション・アドレスの取り出し"のシス
0102 CD0500    CALL   0005H      テム・コール。H, Lレジスタに、ディスク・ド
                                ライブのアロケーション・ベクトルのベース
                                アドレスが得られる。

0105 061F      MVI    B, 31      バイト表示のカウンタとして、アロケーション・ベクトル・エリアの
                                バイト長をセットする。この値はドライブによって異なる。

LOOP:
0107 7E        MOV    A, M      アロケーション・ベクトル・エリアの先頭から、
0108 CD1601    CALL   HEXDSPLY  1バイトずつ16進でコンソールに表示して行
                                く。
010B 3E20      MVI    A, ' '    スペースをコンソールに出力。各バイトの表
010D CD2E01    CALL   CONOUT    示をスペースで区切る。

0110 23        INX    H         H, Lレジスタに、アロケーション・ベクトルの
0111 05        DCR    B         次のバイト・アドレスをセットしてループす
0112 C20701    JNZ    LOOP      る。エリア全体を表示し終わったら、本プ
                                グラムを終了して、OP/Mに戻る。

0115 C9        RET

HEXDSPLY:
0116 F5        PUSH   PSW
0117 0F0F0F0F  RRC! RRC! RRC! RRC
011B CD1F01    CALL   HOUT1
011E F1        POP    PSW
011F E60F      HOUT1: ANI    0FH      Aレジスタの値を、16進でコンソールに表示
0121 C630      ADI    30H          するサブルーチン。
0123 FE3A      CPI    3AH
0125 DA2A01    JC     HOUT2
0128 C607      ADI    7
012A CD2E01    HOUT2: CALL   CONOUT
012D C9        RET

CONOUT:
012E C5E5      PUSH   B! PUSH H
0130 0E02      MVI    C, 2
0132 5F        MOV    E, A      Aレジスタのキャラクタを、コンソールに出
0133 CD0500    CALL   0005H      力するサブルーチン。
0136 E1C1      POP    H! POP B
013B C9        RET

0139          END

```

A>

Figure-2.2.45 ファンクション：27 実習プログラムのPRN形式のソース・リスト。

ファンクション：27 実習プログラムの実行

当プログラムを実行するコマンド形式は一種類であり、次に示します。

27]……ただこれだけ。

実行例として、空のディスクから1Kバイトずつファイルをセーブしながら、アロケーション・ベクトルの内容を表示したり、3つの大きなファイルの内、真中に位置するファイルを削除した場合の様

子などを示しますので、アロケーション・ベクトルの働きがよく理解されるものと思います。実行例を次に示します。

ドライブB：上には空のディスクをセットしておく。

A>B:1ログイン・ディスクをB：にチェンジ。

B>A:STAT *.*ディスクB：上のファイルの最初の状態を調べる。

File Not FoundディスクB：上にはファイルがない。

B>A:27実習プログラムを実行し、ディスクが空の状態のドライブB：のアロケーション・ベクトルをみる。

C0 00
00 00 00 00印の値に注目。

B>SAVE 4 A空のディスクに、1 Kバイトのファイル(ファイル名“A”)をセーブする。

B>A:27実習プログラムを実行し、1 Kバイトのファイルがセーブされた状態のドライブB：のアロケーション・ベクトルをみる。

E0 00
00 00 00 00印の値に注目。

B>SAVE 4 Bさらに1 Kバイトのファイル(ファイル名“B”)をセーブする。合計2 Kバイトがセーブされた。

B>A:27実習プログラムを実行し、合計2 Kバイトのファイルがセーブされた状態のドライブB：のアロケーション・ベクトルをみる。

F0 00
00 00 00 00印の値に注目。

B>SAVE 4 Cさらに1 Kバイトのファイル(ファイル名“C”)をセーブする。合計3 Kバイトがセーブされた。

B>A:27実習プログラムを実行し、合計3 Kバイトのファイルがセーブされた状態のドライブB：のアロケーション・ベクトルをみる。

FB 00
00 00 00 00印の値に注目。

B>SAVE 12 DEF今回は3 Kバイトのファイル(ファイル名“DEF”)をセーブする。合計6 Kバイトがセーブされた。

B>A:27実習プログラムを実行し、合計6 Kバイトのファイルがセーブされた状態のドライブB：のアロケーション・ベクトルをみる。

FF 00
00 00 00 00印の値に注目。

B>SAVE 4 Gさらに1 Kバイトのファイル(ファイル名“G”)をセーブする。合計7 Kバイトがセーブされた。

B>A:27実習プログラムを実行し、合計7 Kバイトのファイルがセーブされた状態のドライブB：のアロケーション・ベクトルをみる。

FF B0 00
00 00 00 00印の値に注目。

B>

Figure-2.2.46 ファンクション：27 実習プログラムの実行例①

ドライブ B: 上に新たな空のディスクをセットしておく。

B>A:STAT *.* / ...ディスクB:上のファイルの状態を調べる。

File Not Found ...ディスクB:上にはファイルがない。

B>SAVE 120 X / ... 空のディスク上に、30Kバイト(注)のファイル(ファイル名"X")をセーブする。

B>SAVE 128 Y / ...さらに32Kバイトのファイル(ファイル名"Y")をセーブする。

B>SAVE 128 Z / ...さらに32Kバイトのファイル(ファイル名"Z")をセーブする。

B>A:STAT *.* / ...現在のディスクB: 上のファイルの状態を調べる。

Recs	Bytes	Ext	Acc
240	30k	2	R/W B:X
256	32k	3	R/W B:Y
256	32k	3	R/W B:Z

} これだけセーブされている。

Bytes Remaining On B: 147k

B>A:27 / ...実習プログラムを実行して、ドライブB: のアロケーション・ベクトルをみる。

FF
00 00 00 00 ~印の値に注目。

B>ERA Y / ...32Kバイトのファイル"Y"を削除する。

B>A:STAT *.* / ...現在のディスクB: 上のファイルの状態を調べる。

Recs	Bytes	Ext	Acc
240	30k	2	R/W B:X
256	32k	3	R/W B:Z

} ファイル"Y"が削除されている。

Bytes Remaining On B: 179k

B>A:27 / ...実習プログラムを実行して、ドライブB: のアロケーション・ベクトルをみる。

FF FF FF FF 00 00 00 00 FF FF FF FF 00 00 00 00 00 00 00 00 00 00
00 00 00 00 ~印の値に注目。ファイル"Y"がセーブされていたエリアが0になっている。

B>

(注) ディレクトリ・エリアとして、すでに2Kバイトが使われているので、30+2=32。

Figure-2.2.47 ファンクション: 27 実習プログラムの実行例②

8 インチの標準ディスクの場合は、アロケーション・ベクトルの1バイトで8Kバイトを表現しますが、その1バイトのみに注目して、表現のしかたを解説しておきます。次の図を見るだけで明らかでしょう。

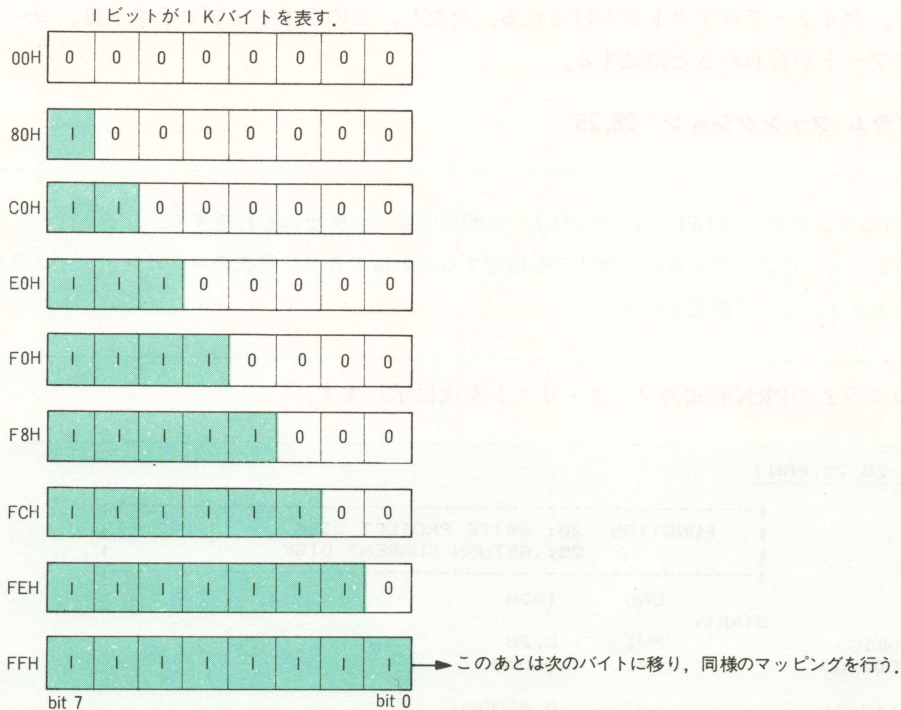


Figure-2.2.48 アロケーション・ベクトル内の1バイト

このように、1ビットが1Kバイトを表し、ビット・パターンが記録状態のマップを表します（0 = 使われていないエリア，1 = 使われているエリア）。

実行例②で～の部分がC0→E0→F0……と変化した理由が理解されたでしょう。

ファンクション：28の実習

ファンクション：28…ディスク・ライト・プロテクトのセット

CALL手順

```

[ MVI    C, 28…… (1CH) ]
[ CALL   0005H ]

```

機能

現在のログイン・ディスクを、書き込み禁止のディスク（リード／オンリー・ディスク）にセットす

る。つまり、ライト・プロテクトがかけられる。ただし、このライト・プロテクトは、コールド・スタートやリブートが行われると消滅する。

実習プログラム ファンクション：28, 25

STATコマンドの、"STAT x:R/O]"に相当するプログラムを作成する。

ただし、x:によって任意のドライブを指定することはできず、現在のログイン・ディスクがリード／オンリーに設定される。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 28-25.PRN]

```

;-----;
; FUNCTION 28: WRITE PROTECT DISK ;
; 25: RETURN CURRENT DISK ;
;-----;

0100          ORG      100H
START:
0100 0E1C      MVI      C,28      } "ライト・プロテクト・ディスク"のシステム・
0102 CD0500    CALL     0005H      } コール。ログイン・ディスク(カレント・ディ
                                } スクが"R/O"となる。

0105 112601    LXI      D,MSGDRV   } "DRIVE..."とコンソールに表示。
0108 CD2001    CALL     MSGOUT

010B 0E19      MVI      C,25      } "カレント・ディスクの取り出し"のシステム・コ
010D CD0500    CALL     0005H      } ール。AレジスタにドライブNo.が得られる。

0110 3C        INR      A          } ドライブNo.をA,B,C,...PのASCIIコードに
0111 F640      ORI      0100*0000B } 変換。
0113 5F        MOV      E,A
0114 0E02      MVI      C,2
0116 CD0500    CALL     0005H      } "コンソールへの出力"のシステム・コールで、
                                } ドライブ名を表示。

0119 112F01    LXI      D,MSGSET   } " : SET R/O"とコンソールに表示。
011C CD2001    CALL     MSGOUT

011F C9        RET      ...CP/Mに戻る。

MSGOUT:
0120 0E09      MVI      C,9
0122 CD0500    CALL     0005H      } 文字列のコンソールへの出力のサブルーチン。
0125 C9        RET

0126 0D0A445249 MSGDRV: DB      0DH,0AH,'DRIVE *'
012F 3A20534554 MSGSET: DB      ': SET R/O',0DH,0AH,'*' } 文字列エリア。

013B          END

```

A>

Figure-2.2.49 ファンクション：28, 25 実習プログラムのPRN形式のソース・リスト。

上記ソース・ファイルからアセンブリ・ソース・ファイルを作り、アセンブルし、“28-25.COM”を生成して実行してみましょう。

ファンクション：28, 25 実習プログラムの実行

当プログラムを実行するコマンド形式を次に示します。

28-25 Jただこれだけ。

上記コマンドを実行することにより、現在のログイン・ディスクがR/Oとなります。実行例を次に示します。

A>STAT J現在のドライブの状態を調べる。

A: R/W, Space: 119K
B: R/W, Space: 79K | A:, B: 共に“R・W”である。

A>28-25 J実習プログラムを実行。

DRIVE A: SET R/Oログイン・ディスクのドライブA: が“R・O”に設定された。

注目: “ログイン・ディスクの取り出し”のシステム・コールを利用した表示。

A>STAT Jドライブの状態を調べる。

A: R/O, Space: 119KA: は“R/O”となっている。
B: R/W, Space: 79KB: は元のまま。

A>DIR TEST28.TXT JドライブA: 上のファイル“TEST28.TXT”の存在を確認する。

A>TEST28 TXT存在している。

A>ERA TEST28.TXT Jファイル“TEST28.TXT”の削除を試みる。

Bdos Err On A: R/O ^Cドライブが“R・O”のため削除できない。Ctrl-Cをキーインし、リブートしてCP/Mに戻す。

A>ERA TEST28.TXT J再度削除を試みる。リブートにより、“R・O”はキャンセルされている。

A>DIR TEST28.TXT Jファイル“TEST28.TXT”の存在を確認する。

NO FILECtrl-Cによるリブートにより、“R・O”がリセットされたので、削除されている。

A>B: Jログイン・ディスクをB: にチェンジ。

B>A: 28-25 J実習プログラムを実行。


```

DRIVE B: SET R/O .....ログイン・ディスクのドライブB:が"R/O"に設定された。
                        注目: "ログイン・ディスクの取り出し"のシステム・コールを利用した表示。

B>A:STAT / .....ドライブの状態を調べる。
A: R/W, Space: 123K .....リポートにより"R/O"→"R/W"に戻っている。
B: R/O, Space: 79K .....今回"R/O"に設定された。

B>^C .....Ctrl-Cによるリポートを行う。
B>A:STAT / .....ドライブの状態を調べる。

A: R/W, Space: 123K .....リポートにより、すべてのドライブの"R/O"はキャンセルされる。
B: R/W, Space: 79K
B>
    
```

Figure-2.2.50 ファンクション：28, 25 実習プログラムの実行。

ファンクション：29の実習

ファンクション：29…リード／オンリー・ベクトルの取り出し

CALL手順

```

    MVI    C, 29..... (=1DH)
    CALL   0005H
    ↓
    ((H, L) レジスタに、R/Oドライブを示すベクトルが格納されている)
    
```

機能

ファンクション：28の“ディスク・ライト・プロテクトのセット”や、STATコマンド、あるいはBDOSにより（例えば、ディスケットを交換して、リポートを行っていない場合など）ライト・プロテクトされたディスク・ドライブのベクトルを報告する。

ベクトルは、ファンクション：24の“ログイン・ベクトルの取り出し”の場合と同様の形式で、(H, L) レジスタに格納される。

実習プログラム ファンクション：29

ライト・プロテクトが付けられたディスク・ドライブのベクトルを表示する。

表示の形式は、ファンクション：24で行ったのと同様に、

$$\begin{array}{cccccccccccccccccccc} & x & x & x & x & x & x & x & x & x & \$ & x & x & x & x & x & x & x & x \\ \uparrow & & & & & & & & & & & & & & & & & & \uparrow \\ \text{ドライブP:} & & & & & & & & & & & & & & & & & & \text{ドライブA:} \end{array}$$

とする。“1”のビットが“R/O”のドライブを示す。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 29.PRN J

```

;-----;
;  FUNCTION  29: GET READ ONLY VECTOR
;-----;

0100          ORG      100H
START:
0100 CD3F01    CALL    CRLFOUT .....復帰・改行を出力。

0103 0E1D      MVI     C,29  "リード/オンリーベクトルの取り出し"のシス
0105 CD0500    CALL    0005H デム・コール。

0108 7C        MOV     A,H
0109 CD1901    CALL    BITOUT  ドライブP: ~I: のビット状態を表示。
010C 1E24      MVI     E,'*'
010E CD3301    CALL    CONOUT  区切りの目印として"$"記号を表示。
0111 7D        MOV     A,L
0112 CD1901    CALL    BITOUT  ドライブH: ~A: のビット状態を表示。

0115 CD3F01    CALL    CRLFOUT .....復帰・改行を出力。

011B C9        RET .....プログラム終了。CP/Mへ戻る。

BITOUT:
0119 0608      MVI     B,B
011B 07        LOOP:   RLC
011C DA2901    JC      OUT1
011F 1E30      MVI     E,'0'
0121 CD3301    CALL    CONOUT
0124 05        DCR     B
0125 C21B01    JNZ     LOOP
0128 C9        RET

OUT1:
0129 1E31      MVI     E,'1'
012B CD3301    CALL    CONOUT
012E 05        DCR     B
012F C21B01    JNZ     LOOP
0132 C9        RET

CONOUT:
0133 F5C5E5    PUSH    PSW! PUSH B! PUSH H
0136 0E02      MVI     C,2
013B CD0500    CALL    0005H
013B E1C1F1    POP     H! POP B! POP I SW
013E C9        RET
Eレジスタのアスキー・コードをコンソール
に出力するサブルーチン

```



```

                                CRLFOUT:
013F 1E0D                      MVI     E,ODH
0141 CD3301                   CALL    CONOUT
0144 1E0A                      MVI     E,0AH
0146 CD3301                   CALL    CONOUT
0149 C9                       RET
                                復帰・改行のサブルーチン.

014A                          END
A>

```

Figure-2.2.51 ファンクション：29 実習プログラムのPRN形式のソース・リスト。

上記ソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルし、“29.COM” を生成して実行してみましょう。

ファンクション：29 実習プログラムの実行

STATコマンドを使ってそれぞれのドライブを“R/O”に設定し、当プログラムで、その結果を調べてみましょう。

当プログラムを実行するコマンド形式は次の通りです。

29 Jただこれだけ。

実行例を次に示します。

```

A>B: J
B>A: J  次のSTATコマンドによる表示が,A: ,B: 共にレポートされるように,例えばこのようにB: にもアクセスしておく.

A>STAT J
A: R/W, Space: 157k
B: R/W, Space: 113k  初期状態の確認.このようにA: ,B: 共に“R/W”である.

A>29 J .....実習プログラムの実行.
00000000*00000000 .....すべて“0”であり,“R/O”のドライブはない.

A>STAT B:=R/O J .....STATコマンドで,ドライブB: を“R/O”に設定する.

A>29 J .....実習プログラムの実行.
00000000*00000010
      ↑
      2番目(ドライブB: )が“R/O”と表示された.

A>STAT J .....STATコマンドで確認してみる.

```



```

A: R/W, Space: 157k | このように、ドライブB: は"R/O"と表示されている。
B: R/O, Space: 113k

A>STAT A:=R/O | さらにSTATコマンドで、ドライブA: も"R/O"に設定する。
A>29 | ..... 実習プログラムの実行。
00000000*00000011
      |
      |----- ドライブA:, B: 共に"R/O"となった。
A>

```

Figure-2.2.52 ファンクション：29 実習プログラムの実行例。

求めるベクトルは、ファンクション：24と同様の形式で、(H, L)レジスタに格納されています。

ファンクション：30の実習

ファンクション：30…ファイル・アトリビュートのセット

CALL手順

```
MVI    C, 30..... (=1EH)
```

```
(D, E) ←FCBアドレス
```

```
CALL    0005H
```



(Aレジスタに、0, 1, 2, 3いずれかのディレクトリ・コード、もしくはファイルが存在しない場合、255 (FFH) が格納されている)

機能

任意のファイルに、リード・オンリー (R/O) や、システム (SYS) ・アトリビュートをセットしたり、それらのリセットである "R/W" や "DIR" アトリビュートにリセットしたりすることができる。FCBの1stファイルのエクステンション用の3バイト (t1, t2, t3, フィールド) の内の2バイトの、それぞれの最上位ビット (bit7) の値により、

t1のbit 7 = 0 のとき "R/W" ファイル

t1のbit 7 = 1 のとき "R/O" ファイル

t2のbit 7 = 0 のとき "SYS" ファイル

t2のbit 7 = 1 のとき "DIR" ファイル

となる (このことからファイル名にカナは使用できない)。

CALL終了後、Aレジスタには0, 1, 2, 3いずれかのディレクトリ・コード, もしくは目的のファイルが存在しなかった場合, 255 (FFH) が格納される。

実習プログラム ファンクション: 30

STATコマンドによる“STAT x:filename.ext \$R/O”などの、ファイル・アトリビュート設定機能に相当するプログラムを作成する。

とりあえず、通常のファイル(“R/W”で“DIR”)に“R/O”, または“SYS”アトリビュートを付けるプログラムの、PRN形式のソース・リストを次に示します。

A>TYPE 30.PRN ↓

```

;-----;
;  FUNCTION 30: SET FILE ATTRIBUTES  ;
;-----;

0100          ORG      100H
005C =        FCB      EQU      005CH

START:
0100 0E09      MVI      C,9
0102 114B01    LXI      D,MSGINP
0105 CD0500    CALL     0005H    "R"または"S"のキーインを促すメッセージを
                                   出力。

KEYIN:
010B 0E01      MVI      C,1
010A CD0500    CALL     0005H    "コンソール入力"のシステム・コール

010D FE52      CPI      'R'
010F CA2101    JZ       SETRO    キー入力が"R"なら"SETRO"へジャンプ。
0112 FE53      CPI      'S'
0114 CA2C01    JZ       SETSYS   キー入力が"S"なら"SETSYS"へジャンプ。

0117 0E02      MVI      C,2
0119 1E3F      MVI      E,'?'
011B CD0500    CALL     0005H    それ以外のキー入力なら"?"を表示して、
                                   再キー入力へ。

011E C30B01    JMP      KEYIN

SETRO:
0121 3A6500    LDA      FCB+9
0124 F6B0      ORI      1000*0000B  FCBのt1のbit7を1にセットし、"FNC30"へ
0126 326500    STA      FCB+9        ジャンプ。"R/O"アトリビュートをセット
0129 C33401    JMP      FNC30        する。

SETSYS:
012C 3A6600    LDA      FCB+10
012F F6B0      ORI      1000*0000B  FCBのt2のbit7を1にセットし、"FNC30"へジ
0131 326600    STA      FCB+10        ャンプ。"SYS"アトリビュートをセットする。

FNC30:
0134 0E1E      MVI      C,30
0136 115C00    LXI      D,FCB
0139 CD0500    CALL     0005H    "ファイル・アトリビュートのセット"のシス
                                   テム・コール。

```



```

013C FEFF      CPI      255  Aレジスタが255なら、ファイルが存在してい
                        なかった。
013E C0         RNZ      Aレジスタが255以外なら、プログラム終了、CP/Mへ戻る。

013F 0E09      MVI      C,9
0141 117901    LXI      D,MSGERR
0144 CD0500    CALL     0005H
0147 C9        RET

014B 0D0A522965 MSGINF: DB      0DH,0AH,'R)ead only or S)ystem ?'
0164 202020494E DB      ' INPUT ( R / S ) >?'
0179 0D0A0D0A46 MSGERR: DB      0DH,0AH,0DH,0AH,'FILE NOT FOUND',0DH,0AH,0DH,0AH,'*'

0190          END

```

A>

Figure-2.2.53 ファンクション：30 実習プログラムのPRN形式のソース・リスト。

上記ソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルし、“31.COM” を生成して実行してみましょう。

ファンクション：30 実習プログラムの実行

当プログラムを実行するコマンド形式を次に示します。

31 □ x : filename . ext] ……この後でRまたはSをキーインする。

このコマンドを実行した後、プログラムの問いに答えて、“R/O” ファイルにするならば“R”をキーイン、“SYS” ファイルにするならば“S”をキーインします。

実行例を次に示します。

A>STAT B:*. *] ……実行前のドライブB：上のファイルの状態を見ておく。

Recs	Bytes	Ext	Acc
60	8k	1 R/W	B: BALANCE.CAL
35	5k	1 R/W	B: BARRIER.CAL
37	5k	1 R/W	B: BRKEVN.CAL
98	13k	1 R/W	B: INSTALL.COM
116	15k	1 R/W	B: INSTALL.DAT
274	35k	3 R/W	B: INSTALL.OVL
2	1k	1 R/W	B: INSTALL.SUB
196	25k	2 R/W	B: SC.COM
164	21k	2 R/W	B: SC.OVL

全部が通常のファイル。
(“R/W”で“DIR”アトリビュート)

Bytes Remaining On B: 113k

A>30 B:SC.COM / ……B：上のファイル“SC.COM”に対して、実習プログラムを起動。

R)ead only or S)ystem ? INPUT (R / S) >R

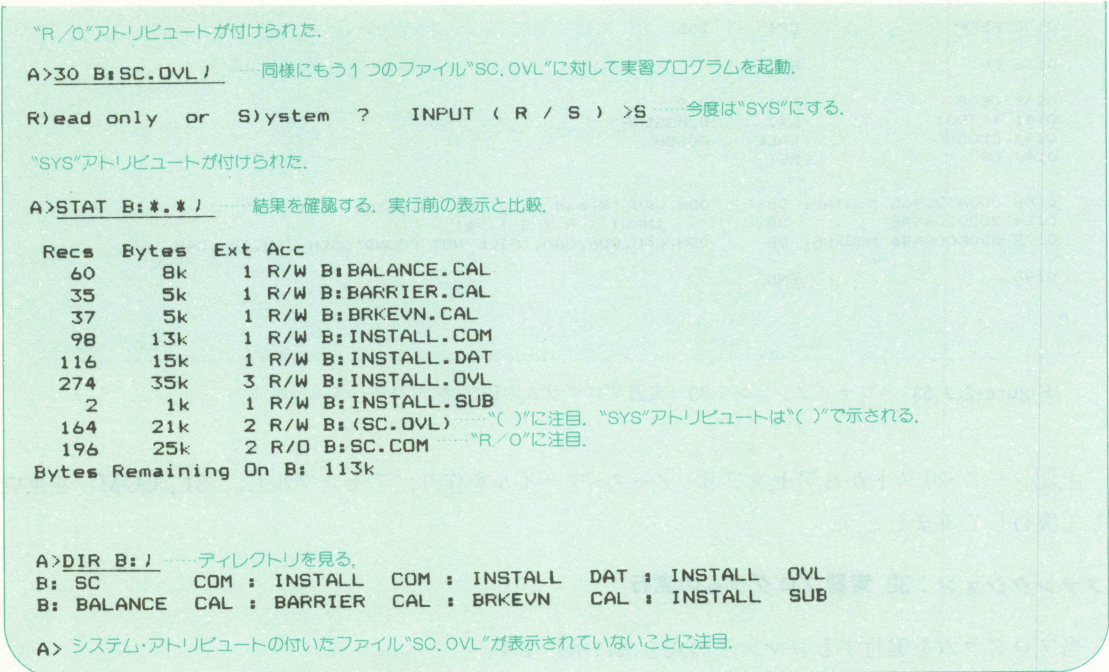


Figure-2.2.54 ファンクション：30 実習プログラムの実行。

当プログラムは、"R/O" と "SYS" アトリビュートを付けるのみですが、これらをリセットする機能も、同様に簡単に追加できます。各自で試みて下さい。

ファンクション：31の実習

ファンクション：31…ディスク・パラメータ・アドレスの取り出し

CALL手順

MVI C, 31..... (=1FH)
CALL 0005H
↓
((H, L) レジスタにディスク・パラメータ・ブロックのベース・アドレスが格納されている)

機能

BIOS先頭の“JUMP ベクトル”に続く、ディスク・パラメータ定義部の“ディスク・パラメータ・ブロック” (15バイト) のベース・アドレスを求め、(H, L) レジスタに格納する。

ユーザー・プログラムにおいて、ディスクの全容量、トラック数、セクタ数、最大ディレクトリ数、その他のディスク諸元を調べたり、この15バイトの値を変更して、例えば1つのドライブを片面使用⇄両面使用をBIOS上で自由に切り替えて使う場合などに利用される (1.1.2～3章参照)。

実習プログラム ファンクション：31

ディスク・パラメータ・ブロックの15バイトをダンプする。

このプログラムのPRN形式のソース・リストを示します。

A>TYPE 31.PRN J

```

;-----;
; FUNCTION 31: GET DISK PARAMETERS ADDRESS ;
;-----;

0100          ORG      100H
START:
0100 0E1F      MVI      C, 31      ; "ディスク・パラメータ・アドレスの取り出し"の
0102 CD0500    CALL     0005H      ; システム・コール。H, Lレジスタに、そのベ
                                ; ス・アドレスが取り出される。

0105 060F      MVI      B, 15      ; 15バイト分ダンプするカウンタをセット。

LOOP:
0107 7E        MOV      A, M
0108 CD1601    CALL     HEXDSPLY   ; 1バイトのダンプ。

010B 3E20      MVI      A, ' '
010D CD2E01    CALL     CONOUT      ; スペースを出力。

0110 23        INX      H
0111 05        DCR      B
0112 C20701    JNZ      LOOP       ; H, Lポインタを1つ進め、次のバイトのダン
                                ; プへ。15バイト出力し終わったら、次の"RET"
                                ; へ。

0115 C9        RET      ; プログラム終了。CP/Mへ戻る。

HEXDSPLY:
0116 F5        PUSH     PSW
0117 0F0F0F0F  RRC! RRC! RRC! RRC
011B CD1F01    CALL     HOUT1
011E F1        POP      PSW
011F E60F      HOUT1: ANI      0FH
                                ; Aレジスタの値を16進でコンソールに表示
                                ; するサブルーチン。
0121 C630      ADI      30H
0123 FE3A      CPI      3AH
0125 DA2A01    JC       HOUT2
0128 C607      ADI      7
012A CD2E01    HOUT2: CALL     CONOUT
012D C9        RET

```



```

CONOUT:
012E C5E5      PUSH B! PUSH H
0130 0E02      MVI     C,2
0132 5F        MOV     E,A
0133 CD0500    CALL    0005H
0136 E1C1      POP H! POP B
0138 C9        RET

0139          END

A>

```

Aレジスタのアスキー・コードを、コンソールに出力するサブルーチン。

Figure-2.2.55 ファンクション：31 実習プログラムのPRN形式のソース・リスト

上記ソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルし、“31.COM” を生成して実行してみましょう。

ファンクション：31 実習プログラムの実行

当プログラムを実行するコマンド形式は単純で、

31 Jただこれだけ。

とキーインするだけで、現在ログインしているドライブのディスク・パラメータ・ブロックの15バイトがダンプされます。

実行例として、8 インチ標準ディスクの場合と NEC PC-8001 CP/M (1W) の場合を次に示します。

A>31 J8インチ標準ディスクの場合の実行例。

1A 00 03 07 00 F2 00 3F 00 C0 00 10 00 02 00

A>

Figure-2.2.56 ファンクション：31 実習プログラムの実行、8 インチ標準ディスクの場合。

A>31 JPC-8001 CP/M(1W)の場合の実行例。

20 00 03 07 00 B3 00 3F 00 C0 00 10 00 02 00

A>

Figure-2.2.57 同上、PC-8001 CP/M (1W) の場合。

Figure-2.2.58 実際のBIOSのテキスト・パラメータ・ブロック部のタイポグラフィの場合。

この161バイトが実習プログラムにのりつけられる。

```

4ABD 1A00
4ABF 03
4A90 07
4A91 00
4A92 F200
4A94 3F00
4A96 00
4A97 00
4A98 1000
4A9A 0200

```

```

;DISK PARAMETER BLOCK, COMMON TO ALL DISKS
;SECTORS PER TRACK      26
;BLOCK SHIFT FACTOR     3
;BLOCK MASK              7
;NULL MASK               0
;DISK SIZE-1            242
;DIRECTORY MAX          63
;ALLOC 0                 192
;ALLOC 1                 0
;CHECK SIZE             16
;TRACK OFFSET           2

```

```

TYPE PCBI0556.PRN) ...-PC-80UM CP/M(1W)のBIOISのチヤノ・ハヤシ・タ・ワロツをサナフオトする。
*** DISK PARAMETER DEFINITION ***
*****
; BASE OF DISK PARAMETER BLOCKS
DPA3 = DPBASE EQU $ XLT0,0000H ; TRANSLATE TABLE
DPA3 00000000 DPE0: DW XLT0,0000H
DPA7 = DABO DPBO EQU $
DPA7 2000 DPA5 03 DPA6 07 DPA7 00 DAB7 00 DAB8 B300 DAB7A 3F00 DAB7C 00 DAB7E 00 DAB0 0200
この151ノットが実習プログラムにおける。

```

同上, PC-8001 CP/M (1W) の場合.

ファンクション：32の実習

ファンクション：32…ユーザー・コードのセット／取り出し

CALL手順

セットの場合	$\left[\begin{array}{l} \text{MVI } C, 32\cdots\cdots (=20\text{H}) \\ (E) \leftarrow \text{ユーザー・コード } (0 \sim 15) \\ \text{CALL } 0005\text{H} \end{array} \right]$
取り出しの場合	$\left[\begin{array}{l} \text{MVI } C, 32 \\ (E) \leftarrow 0\text{FFH} \\ \text{CALL } 0005\text{H} \end{array} \right]$

機能

任意のユーザー・エリアへ移行する。または、現在のユーザー・エリアのNoを報告する。

前者の“セット”の場合は、Eレジスタに任意のユーザーNo(0～15)をセットしてCALLを行う。後者の“取り出し”の場合は、EレジスタにFFHをセットしてCALLを行う。結果はAレジスタに0～15の値として取り出される。

実習プログラム ファンクション：32

ユーザー・エリアを変更する“USER”コマンドと、現在のユーザー・エリアなどを報告する“STAT
USR:”コマンドの一部に相当するプログラムを作成する。
機能は2つであるが、プログラムは一本にまとめる。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 32.PRN↓

```

;-----;
;  FUNCTION  32: SET/GET USER CODE  ;
;-----;

0100      ORG      100H
005C =    FCB      EQU      005CH

```



```

START:
0100 3A5D00      LDA    FCB+1      当プログラムの実行時に、コマンド・ラインの
0103 47          MOV    B,A        パラメータがあるかどうかのチェック。パラ
0104 FE20        CPI    ' '        メータがなければ、"GETUSER"へジャンプ。
0106 CA2201      JZ     GETUSER
0109 3A5E00      LDA    FCB+2      パラメータはあったが、それは2桁であるかど
010C FE20        CPI    ' '        うかのチェック。1桁の場合は"SBYTE"へジャンプ。
010E CA1C01      JZ     SBYTE

0111 E60F        ANI    0000$1111B 2桁のASCII入力を0FHまでのHEX形式に変
0113 C60A        ADI    10          換。

FNC32:
0115 0E20        MVI    C,32      "ユーザー・コードのセット"のシステム・コー
0117 5F          MOV    E,A        ル。
0118 CD0500      CALL   0005H
011B C9          RET              新しいユーザー・エリアに移行した後、プログラ
                                ムを終了してCP/Mに戻る。

SBYTE:
011C 78          MOV    A,B        1桁のASCIIをHEX形式に変換し、
011D E60F        ANI    0000$1111B "FNC32"へジャンプ。
011F C31501      JMP     FNC32

GETUSER:
0122 0E20        MVI    C,32      "ユーザー・コードの取り出し"のシステム
0124 1EFF        MVI    E,0FFH    コール
0126 CD0500      CALL   0005H

0129 CD2D01      CALL   HEXDSPLY 取り出した値を表示。
012C C9          RET              プログラムを終了してCP/Mに戻る。

HEXDSPLY:
012D F5          PUSH   PSW
012E 0F0F0F0F    RRC! RRC! RRC! RRC
0132 CD3601      CALL   HOUT1
0135 F1          POP    PSW
0136 E60F        HOUT1: ANI    0FH      ALレジスタの値を、16進で表示する
0138 C630        ADI    30H      サブルーチン。
013A FE3A        CPI    3AH
013C DA4101      JC     HOUT2
013F C607        ADI    7
0141 CD4501      HOUT2: CALL   CONOUT
0144 C9          RET

CONOUT:
0145 0E02        MVI    C,2      ALレジスタのアスキー・コードをコンソール
0147 5F          MOV    E,A        に表示するサブルーチン。
0148 CD0500      CALL   0005H
014B C9          RET

014C            END

```

A>

Figure-2.2.60 ファンクション：32 実習プログラムのPRN形式のソース・リスト。

ファンクション：32 実習プログラムの実行

当プログラムを実行するコマンド形式を次に示します。

- 32]現在のユーザーNo.を調べる場合。
 32] nユーザー・エリア n を移行する場合。

実行例として、ユーザー・エリア(7)に、ファイルをコピーする例を示します。参考として“USER”コマンド、“STAT”コマンドも使っています。

A>STAT USR:]現状の確認をしておく。

Active User : 0
 Active Files: 0 } 0以外のユーザー・エリアにファイルはない。

A>32]実習プログラムの実行(取り出し)。

00現在のユーザー・エリアは0である。上の結果と同じ。

A>DDT PIP.COM]他のユーザー・エリアにファイルをコピーするために、まずDDTで“PIP.COM”をロード。
 DDT VERS 2.2
 NEXT PC
 1E00 0100
 -^CリポートしてCP/Mに戻る。

A>USER 7]ユーザー・エリア7へ移行。

A>SAVE 29 PIP.COM]そのエリアに“PIP.COM”をセーブ。

A>USER 0]ユーザー・エリアを0に戻す。

A>STAT USR:]現状の確認。

Active User : 0ユーザー・エリアは0。
 Active Files: 0 7ユーザー・エリア0と7にファイルが存在する。

A>32 7]実習プログラムにより、ユーザー・エリア7へ移行。

A>DIR]そのエリアのディレクトリを見る。

A: PIP COM先ほどSAVEした“PIP.COM”のみ存在する。

A>PIP A:=32.COM[00]]PIPパラメータ(00)を使って、当プログラム“32.COM”をユーザー・エリア0→7にコピーする。

A>DIR]その確認

A: PIP COM : 32 COM“32.COM”がコピーされている。

A>32 !このエリアにコピーされた実習プログラムにより、現在のユーザー・エリアを調べる。
07当然ではあるが、エリア7である。

A>32 0 !実習プログラムにより、ユーザー・エリアを0に戻す。

A>32 !その確認。

00ユーザー・エリア0に戻っている。

A>

Figure-2.2.61 ファンクション：32 実習プログラムの実行。

ファンクション：33の実習

ファンクション：33…ランダムな読み出し

CALL手順

MVI C, 33…… (=21H)

(D, E) ←FCBアドレス

CALL 0005H



(読み出しが、正常に行われた場合は、Aレジスタには00が、正常でない場合は、01～06のエラー・コードが格納されている)

機能

(D, E)レジスタでアドレスされるFCBで示されるファイルを、FCBのr0, r1フィールドにセットされているレコード番号に従って、ランダムに読み出す。読み出されたレコード(128バイト)は、DMAバッファに格納される。シーケンシャル・リードの場合と異なり、レコード番号の自動インクリメントは行われない。

レコード番号は2バイトの値であり、LSBはr0, MSBはr1にセットする。フィールドr2には、最初に00をセットしておく。

当ファンクションをCALLするには、まず、目的のファイルをオープンし、r2に00(最初のみ) r0, r1には読み出そうとするレコード番号をセットしてから行う。

読み出しが正常に行われた場合、Aレジスタには00が格納され、正常に行われなかった場合は、01～06のエラー・コードが格納される。エラー・コードの意味を次に示す。

- 00……正常な読み出し。
- 01……ディレクトリ上に登録されていないブロックやロジカル・エクステントから読み出そうとした場合。
- 02……ランダム・モードにならない場合。
- 03……現在のロジカル・エクステントをクローズできない場合。
- 04……登録されていないロジカル・エクステントへシークした場合。
- 05……読み出しモードにならない場合。
- 06……フィジカルなディスクの最終を超えてシークしようとした場合(r2が00でない場合に発生する)。

実習プログラム ファンクション：33

ランダム・リードを行うプログラムを作成する。

ここでは、「ランダム・リードができる」ことの基本的な実習が目的なので、なるべく理解しやすいように、次項の「ランダムな書き込み」での実習プログラムと同じく、アドレス4000Hにあらかじめ格納しておいたレコード番号表に従って、ランダム・リードが行われるプログラムを作成する。リードされたレコードは、順次コンソールに出力されて行く。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 33.PRN /

```

;-----;
;  FUNCTION 33: READ RANDOM  ;
;-----;

0100      ORG      100H
005C =    FCB      EQU      005CH

START:
0100 210040    LXI      H, 4000H } 次に読み出すべきレコードNo. が格納されているアドレスを
0103 228B01    SHLD     BETADR   } 指し示すポインタに、初期値4000Hをストア。

0106 0E0F      MVI      C, 15    }
0108 115C00    LXI      D, FCB   } コマンド・ラインに記述されたファイルのオープン。
010B CD0500    CALL     0005H

010E FEFF      CPI      255      }
0110 CA7101    JZ       ERROR    } 目的のファイルがない場合は"ERROR"へジャンプ。

0113 AF        XRA      A        }
0114 327F00    STA      FCB+35   } "r2"フィールドに00をセット。
    
```


NXTREC:				
0117 2ABB01	LHLD	GETADR		
011A 7E	MOV	A, M		
011B 327D00	STA	FCB+33		
011E 23	INX	H		
011F 7E	MOV	A, M		
0120 327E00	STA	FCB+34		"r0", "r1"にレコードNo. のテーブルからのレコードNo. をセット.
0123 FEFF	CPI	OFFH		
0125 CA0000	JZ	0000H		"r1"がFFHならば、プログラムを終了。 リポートしてCP/Mに戻る.
012B 23	INX	H		
0129 228B01	SHLD	GETADR		次のレコードNo. の格納されているアドレス を、ポインタにセット.
012C 0E21	MVI	C, 33		
012E 115C00	LXI	D, FCB		
0131 CD0500	CALL	0005H		"ランダムな読み出し"のシステム・コール.
0134 B7	ORA	A		
0135 C27101	JNZ	ERROR		正常な読み出しが行われない場合は、 "ERROR"へジャンプ.
013B 218000	LXI	H, 0080H		H, Lレジスタに、DMA/バッファの先頭アド レスをセット.
DISP:				
013B 7E	MOV	A, M		
013C CD5001	CALL	HEXDSPY		
013F 23	INX	H		
アドレス80H~FFHの128バイトの内容を、 コンソールへタイプ・アウトする.				
0140 7D	MOV	A, L		
0141 B7	ORA	A		
0142 C23B01	JNZ	DISP		
0145 0E09	MVI	C, 9		
0147 118401	LXI	D, MSGCRLF		
014A CD0500	CALL	0005H		復帰・改行の出力.
014D C31701	JMP	NXTREC		次のレコードの読み出しへループ.
HEXDSPY:				
0150 F5	PUSH	PSW		
0151 0F0F0F0F	RRC!	RRC! RRC! RRC!		
0155 CD5901	CALL	HOUT1		
0158 F1	POP	PSW		
0159 E60F	ANI	0FH		
015B C630	ADI	30H		Aレジスタの値を、16進で表示する サブルーチン.
015D FE3A	CPI	3AH		
015F DA6401	JC	HOUT2		
0162 C607	ADI	7		
0164 CD6801	CALL	CONOUT		
0167 C9	RET			
CONOUT:				
0168 E5	PUSH	H		
0169 0E02	MVI	C, 2		
016B 5F	MOV	E, A		
016C CD0500	CALL	0005H		Aレジスタのアスキー・コードをコンソール に出力するサブルーチン.
016F E1	POP	H		
0170 C9	RET			


```

                                ERROR:
0171 0E09                      MVI    C,9
0173 117A01                    LXI    D,MSGERR エラー・メッセージの出力サブルーチン.
0176 CD0500                    CALL   0005H
0179 C9                        RET

017A 0D0A455252 MSGERR: DB      0DH,0AH,'ERROR !',0DH,0AH,'$'
0186 0D0A0D0A24 MSGCRLF: DB     0DH,0AH,0DH,0AH,'$'

018B                        GETADR: DS      2

018D                        END

A>

```

Figure-2.2.62 ファンクション：33 実習プログラムのPRN形式のソース・リスト.

上記ソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルし、“33.COM”を生成して実行してみましょう。

ファンクション：33 実習プログラムの実行

当プログラムを実行する前に、まず、ランダム・リードを行うためのレコード番号表を、アドレス4000Hからのエリアにロードしておきます。

レコード番号表がロードできたら、当プログラムを実行します。実行するコマンド形式を次に示します。

33_ x : filename . ext Jドライブ x : 上のファイル “filename . ext” を、アドレス4000Hからのレコード番号表に従ってランダム・リードし、タイプアウトする。

実行例として、次項で解説するファンクション：34「ランダムな書き込み」での実習で作成したランダム・ファイル “TEST.RND” を、読み出して、コンソールにタイプアウトしてみましょう（先に次項を実習して下さい）。

読み出す順序は、書き込みを行ったランダム・レコードと同じ順序で（アドレス4000Hからの内容）で行いますので、タイプアウトは、00, 01,0Cの順に、128バイトずつ行われるはずで

この実行例を次に示します。

A>DDT RND, HEX /ランダム・レコードNo. のテーブルをロード(RND, HEXについては、
DDT VERS 2.2 次項のファンクション：34を参照)。
NEXT PC
401C 0000 当プログラムでは、ランダム・レコードNo. のテーブルは4000H〜に設定してある。
^C DDTを終わる。

```
A>DIR B: / .....読み出そうとしているファイルの確認。
B: TEST          RND
```

[illegible][illegible][illegible][illegible][illegible][illegible][illegible][illegible][illegible][illegible]

[illegible][illegible][illegible]

A>次項で行ったランダムな書き込みと同じ順序で各レコードが読み出されていることが、00→0Cの順序から分かる。

Figure-2.2.63 ファンクション：33 実習プログラムの実行。

ファンクション：34の実習

ファンクション：34…ランダムな書き込み

CALL手順

```
MVI      C, 34..... (=22H)
(D, E) ←FCBアドレス
CALL     0005H
```



(書き込みが正常に行われた場合は、Aレジスタに00が、正常でない場合は、01～06のエラー・コードが格納されている)

機能

DMAバッファ（デフォルトは、80H～FFH）の内容をFCBのr0, r1フィールドにセットされているディスク上のレコード番号に書き込む。

詳細は、「読み出し」が「書き込み」となるだけで、前項のファンクション：33「ランダムな読み出し」と同様である。前項を参照。

ただし、エラー・コード05のみは意味が異なり、ディレクトリがフルとなって、これ以上新しいロジカル・エクステンントを作ることができないことを示す。

実習プログラム ファンクション：34

ランダム・アクセスの基本動作を理解するために、ランダムな任意のレコード番号にデータを書き込んで行き、ランダム・ファイルを作成するプログラムを作成する。

分かり易くするために、ランダムな任意のレコード番号は、アクセスする順に、アドレス4000Hからのエリアに事前にロードしておく。このレコード番号表は、当プログラムでは、何の意味もない適当なレコード番号を並べておくだけであるが、本来のランダム・アクセスを利用したプログラムでは、例えば住所録や、ワードプロセッサの“辞書”などの検索のための“リファレンス・テーブル”（対応表）である。

まず、このレコード番号を並べたテーブルを作る。DDTのSコマンドで、直接書き込んで行っても良いが、見て分かり易いように、エディタとアセンブラを使ってランダムなレコード番号を作成した。

ファイル名を“RND. ASM”としてアセンブルし、できた“RND. HEX”をDDTを使って4000Hにロードします（ORGが4000Hとしてあるので、自動的に4000Hからロードされる）。“RND. ASM”をアセンブルしてできたPRNファイル“RND. PRN”を次に示します。

A>TYPE RND.PRN I

4000		ORG	4000H	DDTで、アドレス4000Hからロードされるように、ORGは4000Hとしておく。
4000	0000	DW	00*00h	00
4002	0100	DW	00*01h	01
4004	0200	DW	00*02h	02
4006	A601	DW	01*A6h	03
4008	A501	DW	01*A5h	04
400A	4701	DW	01*47h	05
400C	1802	DW	02*18h	06
400E	1A02	DW	02*1Ah	07
4010	1702	DW	02*17h	08
4012	A700	DW	00*A7h	09
4014	A500	DW	00*A5h	0A
4016	3000	DW	00*30h	0B
4018	2B00	DW	00*2Bh	0C
401A	FFFF	DW	OFF*FFh	↑ テーブルの終わりを示す。
401C		END		この値をそれぞれのレコード番号に書き込もうとする。

← この、合計13レコードのランダムなレコード番号を、アドレス4000Hからのエリアにロードしておく。

適当なレコードNo.を並べて行く

例えばこれは、レコード番号218H（10進で536）を表す。

Figure-2.2.64 ランダム・レコード表を作るためのアセンブリ・ソース・ファイル。

プログラムは、アドレス4000Hからロードされるこのランダムなレコード番号の順にディスクをアクセスし、DAMバッファ上の128バイトのデータを書き込んで行きます。書き込むデータは、結果の確認がしやすいように1レコード、オール00から始まり、1レコードごとに01, 02, 03とインクリメントして行きます。1レコードの128バイトは、すべて同じ値です。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 34.PRN J

```

-----
FUNCTION 34: WRITE RANDOM
-----

0100      ORG      100H
005C =    FCB      EQU      005CH -----FCBアドレスは、デフォルトの5CHに設定。

START:    LXI      H, 4000H
0103      SHLD     GETADR -----レコード番号表から、次々とレコードNo. を
                                取り出すためのアドレス・ポインタ"GET-
                                ADR"に、初期値4000Hをセット。

0106 AF    XRA      A
0107 327601 STA     WTPATTN -----各レコードに書き込むデータを格納するバ
                                ッファ"WTPATTN"に、初期値"00"をセット。

010A 0E16   MVI      C, 22
010C 115C00 LXI      D, FCB
010F CD0500 CALL     0005H -----FCBに格納されているファイル(コマンド・
                                ラインに記述したファイル)を新たに作成す
                                る。

0112 FEFF   CPI      255
0114 CA4F01 JZ       ERROR -----ディレクトリがいつばいで作成できない場合
                                は、"ERROR"にジャンプ。

0117 AF     XRA      A
0118 327F00 STA     FCB+35 -----FCBの"r2"フィールドに00をセットしてお
                                く。

NEXTREC:   LXI      H, 80H
011B 218000 LDA     WTPATTN
011E 3A7601 MOV      C, A
0121 4F

BUFLOWP:   MOV      M, C
0122 71
0123 23     INX      H
0124 7D     MOV      A, L
0125 B7     ORA      A
0126 C22201 JNZ      BUFLOWP -----デフォルトのDMA/バッファ, 80H~FFHの
                                128/バイトを、書き込みデータ用/バッファ"W-
                                TPATTN"に格納されている1/バイトのデータ
                                でフィルする(うめる)。

0129 0C     INR      C
012A 79     MOV      A, C
012B 327601 STA     WTPATTN -----書き込み用データの値を1つインクリメント
                                し、"WTPATTN"に格納。

PUTROR1:   LHLD     GETADR
012E 2A7401 MOV      A, M
0131 7E     STA     FCB+33
0132 327D00 INX      H
0135 23     MOV      A, M
0136 7E     STA     FCB+34 -----アドレス4000Hからのレコード番号表から、
                                アドレス・ポインタ"GETADR"の指し示すレ
                                コードNo.を取り出し、FCBの"r0", "r1"フ
                                ィールドへセットする。

013A FEFF   CPI      0FFH
013C CA5801 JZ       CLOSE -----"r1"の値がFFHの場合は、レコード番号表の
                                終了を示すので、"CLOSE"へジャンプ。

```


013F 23	INX	H	次のレコードに、アドレス・ポインタをセット。
0140 227401	SHLD	GETADR	
0143 0E22	MVI	C, 34	"ランダムな書き込み"のシステム・コール。
0145 115C00	LXI	D, FCB	
0148 CD0500	CALL	0005H	
014B B7	DRA	A	正常に書き込みが行われた場合は、"NXT-REC"へジャンプし、ループ。
014C CA1B01	JZ	NXTREC	
ERROR:			
014F 0E09	MVI	C, 9	エラー・メッセージを出力し、プログラムを終了してCP/Mへ戻る。
0151 116801	LXI	D, MSGERR	
0154 CD0500	CALL	0005H	
0157 C9	RET		
CLOSE:			
0158 0E10	MVI	C, 16	ファイル・クローズのシステム・コール。 クローズが終わって、ファイルが正常に生成されたら、0Hにジャンプし、リポートしてCP/Mに戻る。クローズできない場合は、"ERR-OR"にジャンプ。
015A 115C00	LXI	D, FCB	
015D CD0500	CALL	0005H	
0160 FEFF	CPI	255	
0162 CA4F01	JZ	ERROR	
0165 C30000	JMP	0000	
0168 0D0A455252	MSGERR: DB	0DH, 0AH, 'ERROR !', 0DH, 0AH, '*'	
0174	GETADR: DS	2	
0176	WTPATTN: DS	1	
0177	END		

A>

Figure-2.2.65 ファンクション：34 実習プログラムのPRN形式のソース・リスト。

上記ソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルし、"34.COM"を生成して実行してみましょう。

ファンクション：34 実習プログラムの実行

新たにフォーマットした空ディスクをドライブB：にセットし、当プログラムを実行してみましょう。

当プログラムの実行には、NECのPC-8001CP/Mを使いました。このCP/Mには、ディスクの"スキュー"がなく、フィジカルなセクタNo順にリード/ライトが行われるので、当プログラムの実行結果を解説する際、ランダム・レコード番号、ディレクトリのデータ、実際のセクタの関係が分かり易くなるので好都合です。

当プログラムの実行には、事前にランダム・レコード番号の表をアドレス4000Hからロードしておきます。

まず、その実行例から示します。


```
A>DDT RND.HEX / ..... "RND.HEX"をDDTを使ってロードし、これをランダム番号表として使用する。
                           ソース・ファイル"RND.ASM"の"ORG"は、4000Hとしてあるので、ランダム番号表は
                           自動的に4000Hからロードされる。

DDT VERS 2.2
NEXT PC
401C 0000
-D4000,402F / ..... ロードされたランダム番号の確認。 0000,0001,0002,01A6,...と、ロードされている。
4000 00 00 01 00 02 00 A6 01 A5 01 47 01 1B 02 1A 02 .....G.....
4010 17 02 A7 00 A5 00 30 00 2B 00 FF FF 00 00 00 00 .....0.+.....
4020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
-^C
A>
```

Figure-2.2.66 ランダム・レコード番号表を、アドレス4000Hからロードしておく。

ランダム・レコード番号表がロードされたので、当プログラムを実行してみましょう。
実行するためのコマンド形式を次に示します。

34┐x : filename.ext / ドライブ x : 上に、ランダム・ファイル "filename . ext" が作られる。

実行例を次に示します。

```
A>34 B:TEST.RND / ..... 実習プログラムの実行。ファイル名"TEST.RND"のランダム・アクセス・ファイル
                           が、ドライブB : 上に作られる。

A>STAT B:TEST.RND / ..... ドライブB : 上に作られた、ランダム・ファイル"TEST.RND"の確認。

Recs  Bytes  Ext Acc
227    8k     5 R/W B:TEST.RND ..... 実際は13レコードのみ書き込みが行われたファイルであるのに、
Bytes Remaining On B: 233k ..... レコード数 : 227, バイト数 : 8k, エクステント数 : 5と表示されていることに注目。

A>
```

Figure-2.2.67 ファンクション : 34 実習プログラムの実行。

ランダム・ファイルが、空のディスク上に作られました。

では、ランダム・ファイルとはどのようなものであるのか、まず「レコード番号——ディレクトリ——実際のトラック・セクタ番号」の関係から解説して行きましょう。

まず、"34.COM"の実行によって作られたランダム・ファイル、"TEST.RND"のディレクトリを、シンクウェア・ラブズ社のセクタ・ディスプレイ・プログラムを使って調べてみます。空のディスク上に作られたファイルなので、8インチ標準ディスクやPC-8001の1W(片面)では、ディレクトリ部の先頭である、トラック02のセクタ01から、"TEST.RND"のディレクトリが記録されているはずです。そのディレクトリ部の様子を次に示します。

A>SECDISJ

```

=====  SECTOR DISPLAY PROGRAM  V1.5  =====
              for PC-8001 CP/M
              copyright by SYNCWARE LABS

```

```

input disk name A - D >B
input track# 00-34 >02
input sector# 01-32 >01
Display,  N)ext sector disp.,  A)uto next,  X) any sector  R)ebot CP/M
input  D,  N,  A,  X,  R,  >D

```

DISK=B TRACK=02 SECTOR=01

B:00	00 54 45 53 54 20 20 20 20 52 4E 44	00 00 00 31	.TEST	RND...1
B:10	02 00 00 00 00 09 0B 00 00 00 00 00	00 00 00 00	
B:20	00 54 45 53 54 20 20 20 20 52 4E 44	03 00 00 27	.TEST	RND...'
B:30	00 00 00 00 03 00 00 00 00 00 00 00	00 00 00 00	
B:40	00 54 45 53 54 20 20 20 20 52 4E 44	02 00 00 4B	.TEST	RND...H
B:50	00 00 00 00 00 00 00 00 00 04 00 00	00 00 00 00	
B:60	00 54 45 53 54 20 20 20 20 52 4E 44	04 00 00 1B	.TEST	RND....
B:70	00 00 06 05 00 00 00 00 00 00 00 00	00 00 00 00	

DISK=B TRACK=02 SECTOR=02

B:00	00 54 45 53 54 20 20 20 20 52 4E 44	01 00 00 2B	.TEST	RND... (
B:10	00 00 00 00 07 00 00 00 00 00 00 00	00 00 00 00	
B:20	E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5	E5 E5 E5 E5	
B:30	E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5	E5 E5 E5 E5	
B:40	E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5	E5 E5 E5 E5	
B:50	E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5	E5 E5 E5 E5	
B:60	E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5	E5 E5 E5 E5	
B:70	E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5	E5 E5 E5 E5	

Figure-2.2.68 ファイル "TEST. RND" のディレクトリ部のセクタ・ダンプ。

実習プログラムによって作られたランダム・ファイル "TEST. RND" によって、5つのロジカル・エクステント（1ロジカル・エクステントは32バイトから成る）が生成されていることに注目して下さい。さらに注目する箇所は、それぞれのエクステントの、

13バイト目……そのエクステントの番号（00～31 10進）。

16バイト目……そのエクステント中で使用されたレコード数（1～128 10進）。

17バイト目～32バイト目……アロケーション・マップ。

であり、それぞれのアロケーション・マップの "○" 印の値と、その位置にも注目して下さい。

次に、このディレクトリの注目すべき部分を、分かり易く書き移した表を示します。それぞれのエクステントの13バイト目（エクステント番号）を左端に、16バイト目（レコード数）を右端に置いて、各エクステントを1行で示してあります。

この表を基に、ランダム・ファイルの詳細な解説に入ります。

まず、8インチ標準ディスクでは、1ロジカル・エクステントで16Kバイトのディスク上のデータを管理することができます。各エクステントに、16バイトのディスク・アロケーション・マップがあり、その1バイトがディスク上の1Kバイトのブロックをマッピング（指し示す）するため、16バイトで16Kバイトのディスク上のエリアを管理できる訳です。

ランダム・ファイルでは、シーケンシャル・ファイルと異なり、ロジカル・エクステントの番号が、00, 01, 02, ……と、順番通りには並びません。Figure-2.2.69の例では、00, 03, 02, 04, 01という順になっています。

また、ディスク・アロケーション・マップ上でも、シーケンシャル・ファイルのように、左から順にマッピングされている訳ではありません。中抜けで、とびとびにマッピングされています。

Figure-2.2.68およびFigure-2.2.69で、4番目のエクステントについて説明しましょう。このエクステントの16バイトは、次のようになっています。

```
00 T E S T _ _ _ _ R N D 04 00 00 18
00 00 06 05 00 00 00 00 00 00 00 00 00 00
```

13バイト目のエクステント番号が“04”であるということは、そのエクステントが $04 \times 128 = 04 \times 80$ (HEX) = 200 (HEX) 台の128個のレコードを管理することを示しています。つまり、レコード番号の200 (HEX) ~ 27F (HEX) を、この4番目のエクステントが管理するのです。

そして、アロケーション・マップの3バイト目の“06”という値は、この3バイト目、つまりレコード番号の210 (HEX) ~ 217 (HEX) の8レコードが、ディスク上の1Kバイト単位の位置を示す、ブロック06 (8インチ標準ディスクでは、1ブロックは8レコード=1Kバイト) に当たっていることを示しています。

アロケーション・マップの4バイト目の“05”は、同じく当ファイルのレコード番号218 (HEX) ~ 21F (HEX) の8レコードが、ディスク上のブロック05に当たっていることを示しています。

ブロック番号は、00から始まり、ディレクトリ・エリアの先頭を00として、8セクタごとに01, 02, ……となります。よって、8インチ標準ディスクやPC-8001CP/M (1W) では、次の表のようになります。

(注)標準ディスクは、この表の値にスキュー変換したものが、物理的なセクタ番号となります。

ブロック番号	通算セクタ番号	実際のトラック、セクタ番号	
		標準ディスク 26セクタ/1トラック (スキュー 変換前)	PC-8001(1W)ディスク 32セクタ/1トラック(スキューなし)
ディレクトリ部	00	01~08	01~08
	01	09~16	09~16
	02	17~24	17~24
	03	25~26 01~06	25~32
ファイル格納部	04	07~14	01~08
	05	15~22	09~16
	06	23~26 01~04	17~24
	07	05~12	25~32
	08	13~20	01~08
	⋮	⋮	⋮

Figure-2.2.70 当例題の実行で書き込まれたレコードのディスク上のアロケーション

例えば、Figure-2.2.69のブロック04は、PC-8001 CP/M(1W) のディスクでは、トラック03のセクタ01~08に当たっていることになります。しかしFigure-2.2.64を見ると、ブロック04のレコード番号140~147の内、147のみしか書き込みを行っていません。よって、トラック03のセクタ08のみにデータ"05"が書き込まれているはずです。

実習プログラムを実行した後、実際にデータが書き込まれたセクタをダンプして次に示します。

```
DISK=B TRACK=02 SECTOR=17
B:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

```
DISK=B TRACK=02 SECTOR=18
B:00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
B:10 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
B:20 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
B:30 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
B:40 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
B:50 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
B:60 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
B:70 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
```

```
DISK=B TRACK=02 SECTOR=19
B:00 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 .....
B:10 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 .....
B:20 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 .....
B:30 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 .....
B:40 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 .....
B:50 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 .....
B:60 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 .....
B:70 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 .....
```

```
DISK=B TRACK=02 SECTOR=30
B:00 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 .....
B:10 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 .....
B:20 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 .....
B:30 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 .....
B:40 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 .....
B:50 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 .....
B:60 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 .....
B:70 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 .....
```

```
DISK=B TRACK=02 SECTOR=31
B:00 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 .....
B:10 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 .....
B:20 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 .....
B:30 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 .....
B:40 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 .....
B:50 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 .....
B:60 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 .....
B:70 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 .....
```


DISK=B TRACK=03 SECTOR=08

```

B:00 05 05 05 05 05 05 05 05 05 05 05 05 05 05
B:10 05 05 05 05 05 05 05 05 05 05 05 05 05 05
B:20 05 05 05 05 05 05 05 05 05 05 05 05 05 05
B:30 05 05 05 05 05 05 05 05 05 05 05 05 05 05
B:40 05 05 05 05 05 05 05 05 05 05 05 05 05 05
B:50 05 05 05 05 05 05 05 05 05 05 05 05 05 05
B:60 05 05 05 05 05 05 05 05 05 05 05 05 05 05
B:70 05 05 05 05 05 05 05 05 05 05 05 05 05 05

```

DISK=B TRACK=03 SECTOR=09

```

B:00 06 06 06 06 06 06 06 06 06 06 06 06 06 06
B:10 06 06 06 06 06 06 06 06 06 06 06 06 06 06
B:20 06 06 06 06 06 06 06 06 06 06 06 06 06 06
B:30 06 06 06 06 06 06 06 06 06 06 06 06 06 06
B:40 06 06 06 06 06 06 06 06 06 06 06 06 06 06
B:50 06 06 06 06 06 06 06 06 06 06 06 06 06 06
B:60 06 06 06 06 06 06 06 06 06 06 06 06 06 06
B:70 06 06 06 06 06 06 06 06 06 06 06 06 06 06

```

DISK=B TRACK=03 SECTOR=11

```

B:00 07 07 07 07 07 07 07 07 07 07 07 07 07 07
B:10 07 07 07 07 07 07 07 07 07 07 07 07 07 07
B:20 07 07 07 07 07 07 07 07 07 07 07 07 07 07
B:30 07 07 07 07 07 07 07 07 07 07 07 07 07 07
B:40 07 07 07 07 07 07 07 07 07 07 07 07 07 07
B:50 07 07 07 07 07 07 07 07 07 07 07 07 07 07
B:60 07 07 07 07 07 07 07 07 07 07 07 07 07 07
B:70 07 07 07 07 07 07 07 07 07 07 07 07 07 07

```

DISK=B TRACK=03 SECTOR=24

```

B:00 08 08 08 08 08 08 08 08 08 08 08 08 08 08
B:10 08 08 08 08 08 08 08 08 08 08 08 08 08 08
B:20 08 08 08 08 08 08 08 08 08 08 08 08 08 08
B:30 08 08 08 08 08 08 08 08 08 08 08 08 08 08
B:40 08 08 08 08 08 08 08 08 08 08 08 08 08 08
B:50 08 08 08 08 08 08 08 08 08 08 08 08 08 08
B:60 08 08 08 08 08 08 08 08 08 08 08 08 08 08
B:70 08 08 08 08 08 08 08 08 08 08 08 08 08 08

```

DISK=B TRACK=03 SECTOR=30

```

B:00 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A
B:10 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A
B:20 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A
B:30 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A
B:40 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A
B:50 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A
B:60 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A
B:70 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A

```

DISK=B TRACK=03 SECTOR=32

```

B:00 09 09 09 09 09 09 09 09 09 09 09 09 09 09
B:10 09 09 09 09 09 09 09 09 09 09 09 09 09 09
B:20 09 09 09 09 09 09 09 09 09 09 09 09 09 09
B:30 09 09 09 09 09 09 09 09 09 09 09 09 09 09
B:40 09 09 09 09 09 09 09 09 09 09 09 09 09 09
B:50 09 09 09 09 09 09 09 09 09 09 09 09 09 09
B:60 09 09 09 09 09 09 09 09 09 09 09 09 09 09
B:70 09 09 09 09 09 09 09 09 09 09 09 09 09 09

```



```

DISK=B  TRACK=04  SECTOR=01
B:00  0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B .....
B:10  0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B .....
B:20  0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B .....
B:30  0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B .....
B:40  0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B .....
B:50  0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B .....
B:60  0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B .....
B:70  0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B .....

DISK=B  TRACK=04  SECTOR=12
B:00  0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C .....
B:10  0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C .....
B:20  0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C .....
B:30  0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C .....
B:40  0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C .....
B:50  0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C .....
B:60  0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C .....
B:70  0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C .....

```

Figure-2.2.71 実習プログラムの実行により、書き込みが行われたセクタのダンプ。

- Figure-2.2.64のレコード番号と、書き込むデータのリスト。
- Figure-2.2.68, Figure-2.2.69の作成されたファイルのディレクトリのダンプ・リスト。
- Figure-2.2.70のブロック番号に対するディスク上のセクタ表。
- Figure-2.2.71の実際に書き込みが行われたセクタのダンプ・リスト。

これらの資料を関連付けて見ることにより、ランダム・アクセスの機能が理解できるものと思います。

誤解しやすいものの1つに1レコード番号と、ディスク上のアロケーションの関係があります。

1つのファイル内では、各レコード番号は、ディスク上の該当するセクタと1対1に対応しますが、別のファイルでは、先程のファイルと同じレコード番号でも、全く別のセクタに当たります。つまり、レコード番号は、ディスク上の絶対的な位置を示すものではなく、それぞれのファイル内での相対的な位置を示すものであると理解して下さい。

この意味の理解を助けるために、実習プログラム "34.COM" を、先程のディスク上に前回と同じレコード番号表により、続けてもう一度実行した場合（今度のファイル名は "2ND.RND" とした）のディレクトリ部をダンプして示します。全く同じランダム・レコード番号に書き込みを行った2つのファイルという点に注目して下さい。

DISK=B TRACK=02 SECTOR=01																
B:00	00	54	45	53	54	20	20	20	20	52	4E	44	00	00	31	.TEST RND...1
B:10	02	00	00	00	00	09	0B	00	00	00	00	00	00	00	00
B:20	00	54	45	53	54	20	20	20	20	52	4E	44	03	00	27	.TEST RND...'
B:30	00	00	00	00	03	00	00	00	00	00	00	00	00	00	00
B:40	00	54	45	53	54	20	20	20	20	52	4E	44	02	00	48	.TEST RND...H
B:50	00	00	00	00	00	00	00	00	00	04	00	00	00	00	00
B:60	00	54	45	53	54	20	20	20	20	52	4E	44	04	00	1B	.TEST RND....
B:70	00	00	06	05	00	00	00	00	00	00	00	00	00	00	00

DISK=B TRACK=02 SECTOR=02																
B:00	00	54	45	53	54	20	20	20	20	52	4E	44	01	00	28	.TEST RND...(
B:10	00	00	00	00	07	00	00	00	00	00	00	00	00	00	00
B:20	00	32	4E	44	20	20	20	20	20	52	4E	44	00	00	31	.2ND RND...1
B:30	0A	00	00	00	00	11	10	00	00	00	00	00	00	00	00
B:40	00	32	4E	44	20	20	20	20	20	52	4E	44	03	00	27	.2ND RND...'
B:50	00	00	00	00	0B	00	00	00	00	00	00	00	00	00	00
B:60	00	32	4E	44	20	20	20	20	20	52	4E	44	02	00	48	.2ND RND...H
B:70	00	00	00	00	00	00	00	00	00	0C	00	00	00	00	00

DISK=B TRACK=02 SECTOR=03																
B:00	00	32	4E	44	20	20	20	20	20	52	4E	44	04	00	1B	.2ND RND....
B:10	00	00	0E	0D	00	00	00	00	00	00	00	00	00	00	00
B:20	00	32	4E	44	20	20	20	20	20	52	4E	44	01	00	28	.2ND RND...(
B:30	00	00	00	00	0F	00	00	00	00	00	00	00	00	00	00
B:40	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5
B:50	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5
B:60	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5
B:70	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5

Figure-2.2.72 全く同じランダム・レコード番号表により、ファイル名の異なる2つのファイルを作る。

前回の“TEST . RND”に続いて、同じようにディレクトリが作られました。しかし、アロケーション・マップのそれぞれのブロック番号だけが異なっており、同じレコード番号を使っても書き込みが行われた物理的なディスク上の位置が異なっていることが分かります。

では次に、ランダムな13のレコードで作られた2つのファイル、“TEST . RND”と“2ND . RND”の内容（ランダム・レコードの取り出し順に、それぞれのセクタが、128バイトの00, 01, 02, …… , 00でフィルされている）と、それらのデータが書き込まれたディスク上の物理的な位置の関係を示します。

データが書き込まれたセクタの位置は、Figure-2.2.72と、Figure-2.2.70からも求めることができますが、ここでは、フォーマットした直後の完全に空のディスクと、トラックtoトラックの実比較によって調べてみます。先程のシンクウェア・ラブズ社のセクタ・ディスプレイと同じパッケージに含まれている、コンペア・ディスクのプログラムを使っています。

ランダム・アクセスにより書き込まれた2つのファイルが占めるセクタは、“COMPARE ERROR”となって、すべて表示されています。それらのセクタに何が書き込まれていたかを、その右に示しました。

A>CMPDISK I

```

=====  DISK COMPARE PROGRAM  V1.5  =====
          FOR PC-8001 CP/M
          copyright by  -SYNCWARE LABS-

```

input start track# 00-34 >02

input start sector# 01-32 >17

```

COMPARE ERROR ON TRACK= 02  SECTOR= 17 ----00
COMPARE ERROR ON TRACK= 02  SECTOR= 18 ----01
COMPARE ERROR ON TRACK= 02  SECTOR= 19 ----02
COMPARE ERROR ON TRACK= 02  SECTOR= 30 ----04
COMPARE ERROR ON TRACK= 02  SECTOR= 31 ----03

```

```

COMPARE ERROR ON TRACK= 03  SECTOR= 08 ----05
COMPARE ERROR ON TRACK= 03  SECTOR= 09 ----06
COMPARE ERROR ON TRACK= 03  SECTOR= 11 ----07
COMPARE ERROR ON TRACK= 03  SECTOR= 24 ----08
COMPARE ERROR ON TRACK= 03  SECTOR= 30 ----0A
COMPARE ERROR ON TRACK= 03  SECTOR= 32 ----09

```

ファイル
TEST. RND

```

COMPARE ERROR ON TRACK= 04  SECTOR= 01 ----0B
COMPARE ERROR ON TRACK= 04  SECTOR= 12 ----0C
COMPARE ERROR ON TRACK= 04  SECTOR= 17 ----00
COMPARE ERROR ON TRACK= 04  SECTOR= 18 ----01
COMPARE ERROR ON TRACK= 04  SECTOR= 19 ----02
COMPARE ERROR ON TRACK= 04  SECTOR= 30 ----04
COMPARE ERROR ON TRACK= 04  SECTOR= 31 ----03

```

```

COMPARE ERROR ON TRACK= 05  SECTOR= 08 ----05
COMPARE ERROR ON TRACK= 05  SECTOR= 09 ----06
COMPARE ERROR ON TRACK= 05  SECTOR= 11 ----07
COMPARE ERROR ON TRACK= 05  SECTOR= 24 ----08
COMPARE ERROR ON TRACK= 05  SECTOR= 30 ----0A
COMPARE ERROR ON TRACK= 05  SECTOR= 32 ----09

```

ファイル
2ND. RND

```

COMPARE ERROR ON TRACK= 06  SECTOR= 01 ----0B
COMPARE ERROR ON TRACK= 06  SECTOR= 12 ----0C

```

```

+++++ END OF LAST TRACK +++++
!!!!!! COMPARE ERROR EXIST !!!!!

```

↑
それぞれのセクタに書
き込まれているデータ

Figure-2.2.73 同一ランダム・レコード番号で2つのファイルを作成した場合の、書き込まれた内容とディスク上の実セクタの関係。

ランダム・アクセスの基本的な機能は、ファンクション：33, 34, 36, 40の実習プログラムで解説されていますが、要するに、「レコードのランダムな読み書きができる」という一言に尽きます。しかし我々CP/Mユーザーにとって、問題はその後のものであり、これらの機能をいかに有効に応用するかということでしょう。

漢字ワード・プロセッサなら「辞書」の検索に応用できるでしょう。「辞書」の“早見表”をシーケンシャル・ファイルで作り、“早見表”で目標を一応見つけてから、「辞書」にランダム・アクセスさせるか、あるいはもっと効率の良いテクニックを使うのか……。

ランダム・ファイルの応用には、様々なテクニックがあるでしょう。そこは各自で試みて頂きたいと思います。

ファンクション：35の実習

ファンクション：35…ファイル・サイズの計算

CALL手順

```

[ MVI    C, 35…… (=23H)
  (D, E) ←FCBアドレス
  CALL    0005H
  ↓
  (FCBのr0, r1, r2にファイル・サイズがセットされる) ]

```

機能

(D, E)レジスタでアドレスされるFCBで示されるファイルの、エンド・オブ・ファイルの次のレコード番号を、FCBのr0, r1フィールドにセットする。LSBはr0に、MSBはr1にセットされる。ファイルが最大のレコード数である65536を持っている場合は、フィールドr2に01がセットされる。この機能により、ファイルの大きさ（総レコード数）を知ることができる。

FCBのフィールドr0, r1にセットされた値は、シーケンシャル・ファイルの場合は、実際のファイルのサイズと一致するが、ランダム・ファイルの場合は、ファンクション：35の実習でも経験したように“中ぬけ”が存在するために、実際に書き込みが行われているレコード数より、ずっと多い見かけ上のファイル・サイズとなる。

もう1つの機能（機能と言うより、上記機能に他ならないが）は、当ファンクションを実行することにより、ファイルの最終レコードの次がFCBのr0, r1にセットされることを利用して、続けてファンクション：34を実行することにより、そのレコード位置にランダムな書き込みを行うことができる。この機能の様々な応用が考えられる。

実習プログラム ファンクション：35

当ファンクションを実行した後のFCBのr0, r1, r2の各フィールドを、ただ表示するだけのプログラムを作り、当機能を確認する。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 35.PRN I

```

;-----;
;  FUNCTION 35: COMPUTE FILE SIZE  ;
;-----;

0100      ORG      100H
005C =    FCB      EQU      005CH -----FCBアドレスはデフォルトの5CHに設定.

START:
0100 0E23      MVI      C,35
0102 115C00     LXI      D,FCB
0105 CD0500     CALL     0005H  "ファイル・サイズの計算"のシステム・コール

010B 3A7D00     LDA      FCB+33
010B CD1B01     CALL     DATAOUT
010E 3A7E00     LDA      FCB+34
0111 CD1B01     CALL     DATAOUT  FCBのr0, r1, r2フィールドの順に, その値を
                                   コンソールに表示する.

0114 3A7F00     LDA      FCB+35
0117 CD1B01     CALL     DATAOUT

011A C9         RET -----プログラムを終了して, CP/Mに戻る.

DATAOUT:
011B F5         PUSH     PSW
011C 3E20       MVI      A,' '
011E CD3E01     CALL     CONOUT  ALレジスタの値を, スペースを前置きして,
0121 F1         POP      PSW      コンソールに16進で表示するサブルーチン.
0122 CD2601     CALL     HEXDSPLY
0125 C9         RET

HEXDSPLY:
0126 F5         PUSH     PSW
0127 0F0F0F0F   RRC! RRC! RRC! RRC
012B CD2F01     CALL     HOUT1
012E F1         POP      PSW
012F E60F       ANI      0FH
0131 C630       ADI      30H
0133 FE3A       CPI      3AH
0135 DA3A01     JC       HOUT2
013B C607       ADI      7
013A CD3E01     HOUT2:  CALL     CONOUT
013D C9         RET

CONOUT:
013E 0E02       MVI      C,2
0140 5F         MOV      E,A
0141 CD0500     CALL     0005H  ALレジスタのアスキー・キャラクタを, コンソールに出力するサブルーチン.
0144 C9         RET

0145          END

```

A>

Figure-2.2.74 ファンクション：35 実習プログラムのPRN形式のソース・リスト。

上記ソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルし、“35.COM” を生成して実行してみましょう。

ファンクション：35 実習プログラムの実行

実行するためのコマンド形式を次に示します。

35 x : filename . ext **]** ドライブ x : 上のファイル “filename . ext” のファイル・サイズを示すFCBのr0, r1, r2の値を表示する。

実行例として、シーケンシャル・ファイルの“PIP.COM” と、ファンクション：34の実習で作成したランダム・ファイル“TEST.RND” に対して実行したものを次に示します。

A>STAT PIP.COM] ファイル“PIP.COM”の状態を調べておく。

Recs	Bytes	Ext	Acc
58	8k	1 R/W	A: PIP.COM

Bytes Remaining On A: 125k

..... Recsの項の値“58”(HEXでは3A)に注目。

A>35 PIP.COM] ファイル“PIP.COM”に対して、実習プログラムの実行。

3A	00	00
r0	r1	r2	ファイル・サイズは003A(HEX)と表示された。

STATコマンドでの値と一致する。

A>STAT B: TEST.RND] ランダム・ファイル“TEST.RND”の状態を調べておく。

Recs	Bytes	Ext	Acc
227	8k	5 R/W	B: TEST.RND

Bytes Remaining On B: 233k

..... Recsの項の値“227”(HEXでは021B)に注目。

A>35 B: TEST.RND] ファイル“TEST.RND”に対して、実習プログラムの実行。

1B 02 00 ファイル・サイズは021B(HEX)と表示された。STATコマンドでの値と一致する。

A>

Figure-2.2.75 ファンクション：35 実習プログラムの実行例。

ファンクション：36の実習

ファンクション：36……ランダム・レコードのセット

CALL手順

```
MVI    C, 36…… (=24H)
```

```
(D, E) ←FCBアドレス
```

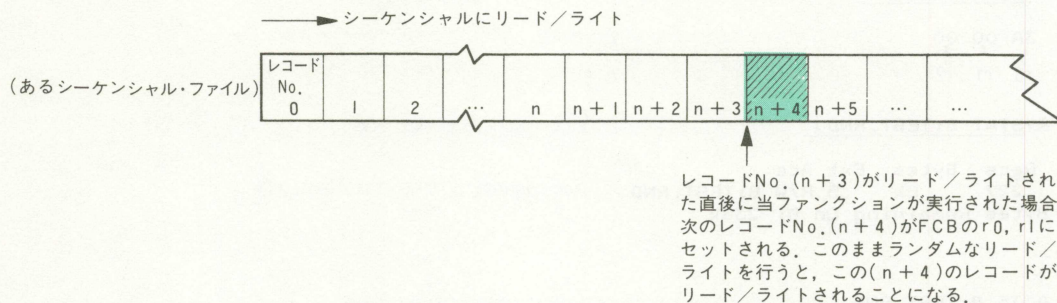
```
CALL   0005H
```



```
(FCBのr0, r1, r2に、ランダム・レコードNo.をセットする)
```

機能

ファイルをシーケンシャルにリード／ライトして、任意の位置で当ファンクションを実行すると、シーケンシャルにリード／ライトした最後のレコード+1のレコードNo.をFCBのr0, r1, r2フィールドにセットし、そのレコードからのランダムなリード／ライトを可能にする。



実習プログラム ファンクション：36

当ファンクションを基にしたランダム・アクセスによる検索プログラムの作成例が、次の2.3章にあります。その検索プログラムでは、シーケンシャルにリード／ライトされる部分が、各インデックスに対応したランダム・レコードNo.を格納するマップ部となっています。ただし、ここでのプログラムではマップ部は、わずか1レコードです。1レコードですが、このマップ部を実習のために一応シーケンシャルにリード／ライトを行っています。

当ファンクションを含めて、ランダム・アクセスの応用例として、2.3章の「ランダム・アクセスによる検索プログラムの作成」を参照して下さい。

ファンクション：37の実習

ファンクション：37…ディスク・ドライブのリセット

CALL手順

```

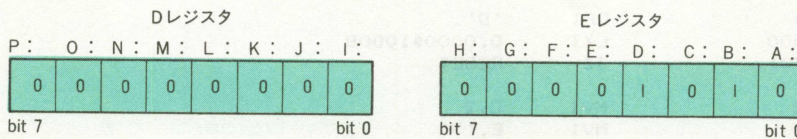
MVI    C, 37…… (=25H)
(D, E) ← ドライブ・ベクトル
CALL   0005H
      ↓
((D, E) レジスタの16ビットのベクトルで示されるドライブがリセットされる)

```

機能

(D, E)レジスタにセットする16ビットのベクトルに従って、ディスク・ドライブのA：～P：の任意のものをリセット状態にする。

例えば、16ビットのベクトルが次のようであれば、



上記ベクトルにより、ドライブB：とD：がリセットされる。

ディスク・ドライブは、リセットによりオンライン状態を解かれ、ドライブがR/Oにセットされている場合は、R/Wにリセットされる。

実習プログラム ファンクション：37

当ファンクションは、一度に複数のドライブをリセットできるが、ここでは実習プログラムを簡単にするため、A：～D：の4つのドライブの内、任意の1つのドライブをリセットするプログラムを作成する。

プログラムを実行すると、リセットしたいドライブ名を問い合わせますので、A～Dの任意のドライブ名をキーインすると、そのドライブがリセットされます。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 37.PRN /

```

;-----;
;  FUNCTION 37: RESET VRIVE (Vector)
;-----;

0100          ORG      100H
START:
0100 0E09      MVI      C,9
0102 113D01    LXI      D,MSGKIN
0105 CD0500    CALL     0005H    ;ドライブ名のキーインを促すメッセージを出力.

KINERR:
0108 0E01      MVI      C,1
010A CD0500    CALL     0005H    ;コンソール入力のシステム・コール.

010D FE41      CPI      'A'
010F 110100    LXI      D,0000*0001B
0112 CA3701    JZ       RESET    ;キー入力が"A"なら、DEレジスタに01をセットして"RESET"へジャンプ.

0115 FE42      CPI      'B'
0117 110200    LXI      D,0000*0010B
011A CA3701    JZ       RESET    ;キー入力が"B"なら、DEレジスタに02をセットして"RESET"へジャンプ.

011D FE43      CPI      'C'
011F 110400    LXI      D,0000*0100B
0122 CA3701    JZ       RESET    ;キー入力が"C"なら、DEレジスタに04をセットして"RESET"へジャンプ.

0125 FE44      CPI      'D'
0127 110800    LXI      D,0000*1000B
012A CA3701    JZ       RESET    ;キー入力が"D"なら、DEレジスタに08をセットして"RESET"へジャンプ.

012D 0E02      MVI      C,2
012F 1E3F      MVI      E,'?'
0131 CD0500    CALL     0005H    ;キー入力がA~D以外の場合は、"?"を出力して、再びキー入力に帰る.
0134 C30B01    JMP      KINERR

RESET:
0137 0E25      MVI      C,37
0139 CD0500    (LXI D,Vector) CALL 0005H ;"ディスク・ドライブのリセット"のシステム・コール.
;ここに来る以前に、DEレジスタのベクトルはセットされている.

013C C9        RET      ;プログラムを終了してCP/Mに戻る.

013D 0D0A494E50 MSGKIN: DB      0DH,0AH,'INPUT DRIVE NAME TO BE RESET >*'

015F          END

```

Figure-2.2.76 ファンクション：37 実習プログラムのPRN形式のソース・リスト。

上記ソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルし、“37.COM”を生成して実行してみましょう。

ファンクション：37 実習プログラムの実行

当プログラムの実行は、まず、

37]

とキーインし、ドライブ名を問い合わせるメッセージに従って、A～Dまで（本来はA～Pの16ドライブを任意にリセット可能であるが、当プログラムではA～Dの4つのドライブに制限してある）のドライブ名をキーインすることにより行われます。

実行例を次に示します。

A>STAT A:=R/O /ドライブA：をR/Oにセット。

A>STAT B:=R/O /ドライブB：もR/Oにセット。

A>STAT /その確認。

A: R/O, Space: 128k

B: R/O, Space: 233k

} A:, B: 共にR/Oとなっている。

A>37 /実習プログラムの実行。

INPUT DRIVE NAME TO BE RESET >AドライブA：をリセット。

A>STAT /結果の確認。

B: R/O, Space: 233k

.....ドライブA：がリセットされた直後のSTATコマンドの実行なので、ドライブA：のレポートが行われていない。B：は元のままのR/Oである。

A>37 /再度、実習プログラムの実行。

INPUT DRIVE NAME TO BE RESET >B今度はドライブB：をリセット。

A>STAT /結果の確認。

A: R/W, Space: 128k

.....ドライブB：はリセットされて、OFF LINE状態であるので、レポートされていない。当然R/Wにリセットされている。先程のA：のリセット実行で、A：がR/Wにリセットされている。

A>

Figure-2.2.77 ファンクション：37 実習プログラムの実行例。

ファンクション：40の実習

ファンクション：40…ゼロ書き込み(ゼロ・フィル)を伴うランダムな書き込み

CALL手順

```
MVI    C, 40…… (=28H)
      (D, E) ←FCBアドレス
CALL    0005H
```



(書き込みが正常に行われた場合はAレジスタに00が、正常でない場合は01～06のエラー・コードが格納される)

機能

ファンクション：34の「ランダムな書き込み」と同じであるが、当ファンクションは、データが実際には書き込まれない“ホール”（中ぬけ）となるレコードも含めて、書き込みの対象となるブロックのすべてを事前に00でうめておき、その上でデータの書き込みを行う——という機能が追加されている。その他は、ファンクション：34と全く同一なので、その項を参照。

実習プログラム ファンクション：40

ファンクション：34の実習プログラム (Figure-2.2.65) の、ファンクション：34のシステム・コールの部分をも、ファンクション：40に変えただけで、そっくり同じものです。詳細はファンクション：34の項を参照下さい。

このプログラムのPRN形式のソース・リストを次に示します。

A>TYPE 40.PRN /

ファンクション：34と同一プログラム。 部が異なるのみ。

```
-----;
; FUNCTION 40: WRITE RANDOM WITH ZERO FILL
;-----;
```

```
0100      ORG      100H
005C =    FCB      EQU      005CH

      START:
0100 210040    LXI      H, 4000H
0103 227401    SHLD     GETADR

0106 AF       XRA      A
0107 327601    STA      WTPATTN
```



```

010A 0E16      MVI    C,22
010C 115C00    LXI    D,FCB
010F CD0500    CALL   0005H

0112 FEFF      CPI    255
0114 CA4F01    JZ     ERROR

0117 AF        XRA     A
0118 327F00    STA     FCB+35

NXTREC:
011B 218000    LXI    H,80H
011E 3A7601    LDA     WTPATTN
0121 4F        MOV     C,A

BUFLOOP:
0122 71        MOV     M,C
0123 23        INX     H
0124 7D        MOV     A,L
0125 B7        ORA     A
0126 C22201    JNZ     BUFLOOP

0129 0C        INR     C
012A 79        MOV     A,C
012B 327601    STA     WTPATTN

PUTROR1:
012E 2A7401    LHLD   GETADR
0131 7E        MOV     A,M
0132 327D00    STA     FCB+33
0135 23        INX     H
0136 7E        MOV     A,M
0137 327E00    STA     FCB+34

013A FEFF      CPI    0FFH
013C CA5801    JZ     CLOSE

013F 23        INX     H
0140 227401    SHLD   GETADR

0143 0E28      MVI    C,40 .....ここが異なるだけ.
0145 115C00    LXI    D,FCB
0148 CD0500    CALL   0005H

014B B7        ORA     A
014C CA1B01    JZ     NXTREC

ERROR:
014F 0E09      MVI    C,9
0151 116801    LXI    D,MSGERR
0154 CD0500    CALL   0005H
0157 C9        RET

CLOSE:
0158 0E10      MVI    C,16
015A 115C00    LXI    D,FCB
015D CD0500    CALL   0005H

0160 FEFF      CPI    255
0162 CA4F01    JZ     ERROR
0165 C30000    JMP     0000

0168 0D0A455252 MSGERR: DB    0DH,0AH,'ERROR !',0DH,0AH,'*'

```



```

0174          GETADR: DS      2
0176          WTPATTN: DS    1

0177          END
A>

```

Figure-2.2.78 ファンクション：40 実習プログラムのPRN形式のソース・リスト.

上記ソース・リストからアセンブリ・ソース・ファイルを作り、アセンブルし、“40.COM” を生成して実行してみましょう。

ファンクション：40 実習プログラムの実行

当プログラムの実行方法と実行結果は、ファンクション：34と同じです。ただし、実際のデータが書き込まれているレコード以外のブロックには、すべてに00が書き込まれているはずです。

実行例を次に示します。

```

A>DDT RND.HEX / .....ランダム・レコード番号表をアドレス4000Hからロード。
DDT VERS 2.2
NEXT PC
401C 0000
-^C .....CP/Mに戻る。

A>40 B:TEST40.RND / .....ドライブB:上に、ランダム・ファイル“TEST40.RND”を作る。
A>

```

Figure-2.2.79 ファンクション：40 実習プログラムの実行.

実行が終わり、ランダム・ファイル“TEST40.RND”が作られました。データが書き込まれていない部分に、00が書かれているかどうか確認してみましょう。

ファンクション：34の項のFigure-2.2.72を参照して下さい。この表の“TEST.RND”の部分が、今回の“TEST40.RND”の部分です。このトラック02のセクタ17から、トラック00のセクタ16（データの書き込みが行われたのは、セクタ12までであるが、ブロック単位ではセクタ16まで）までの“中ぬけ”の部分をセクタ・ダンプ・プログラムで調べた結果、すべてに00が記録されていました。

そのファイルの最終ブロックのセクタ・ダンプを次に示します。

これ以前で、データが書き込まれていない、すべてのブロックには、00が書かれている。

このように00が書かれている。

DISK=B TRACK=04 SECTOR=16

```

B:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
B:70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

ファイル“TEST40.RND”の最終ブロック。

DISK=B TRACK=04 SECTOR=17

```

B:00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B:10 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B:20 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B:30 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B:40 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B:50 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B:60 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B:70 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

```

- ・ フォーマットしたままの、全く手が付けられていないエリア。
(PC-8031ミニディスクは“FF”でイニシャライズされる)

Figure-2.2.80 ファンクション：40 による00の書き込みの確認。

ランダム・アクセスに関しては、スキュー・ファクタのないPC-8001 CP/Mを使って、セクタ番号との関連を理解し易いよう配慮しましたが、スキューを持つCP/Mでは、それぞれのスキュー・ファクタでセクタ番号が変換されていますので換算しなければなりません。ご注意下さい。

2.3 ランダム・アクセスによる検索プログラムの作成

今まで解説を行ってきた“システム・コール”を総合する意味で、ランダム・アクセスを利用した、“電子早見帳”とでも言うべき、検索プログラムを作ってみましょう。

ここで紹介するプログラムは、筆者が「応用CP/M」向けに、学習用としてできるだけ簡素にデザインしたものであり、内容は、“電子早見帳”としての骨子に近い基本的な機能を持っているに過ぎません。しかし、このようなプログラムでも、それを作成するには、CP/Mに関する広い知識と実際にCP/Mを操作して開発を行う経験が必要でしょう。この2.3章は、言わばCP/M全般の基礎講座の卒業制作のようなものです。

2.3.1 プログラムの仕様

- プログラム名……電子早見帳。
- 対象機種……すべてのCP/Mマシン。

- 機能……………インデックス（見出し）として、アルファベット、数字、記号、カナなどの1文字を指定すると、それに対応する項目のデータがスクリーンに表示される。項目データの入力は、インデックスの1文字を指定した後、続けてそのデータを入力することにより行われる。
- 収容可能項目数…42項目
- 1項目当りの文字数…128文字以内

仕様は、上記のようなものです。

各項目の構成は、文字または記号の1文字と128文字以内の自由なエリアから成っています。図で示すと次のようになります。

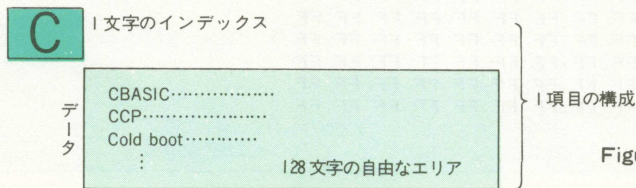


Figure-2.3.1 1項目内のデータ並び

電子早見帳プログラムの考え方

当プログラムのアルゴリズムを図で解説しましょう。

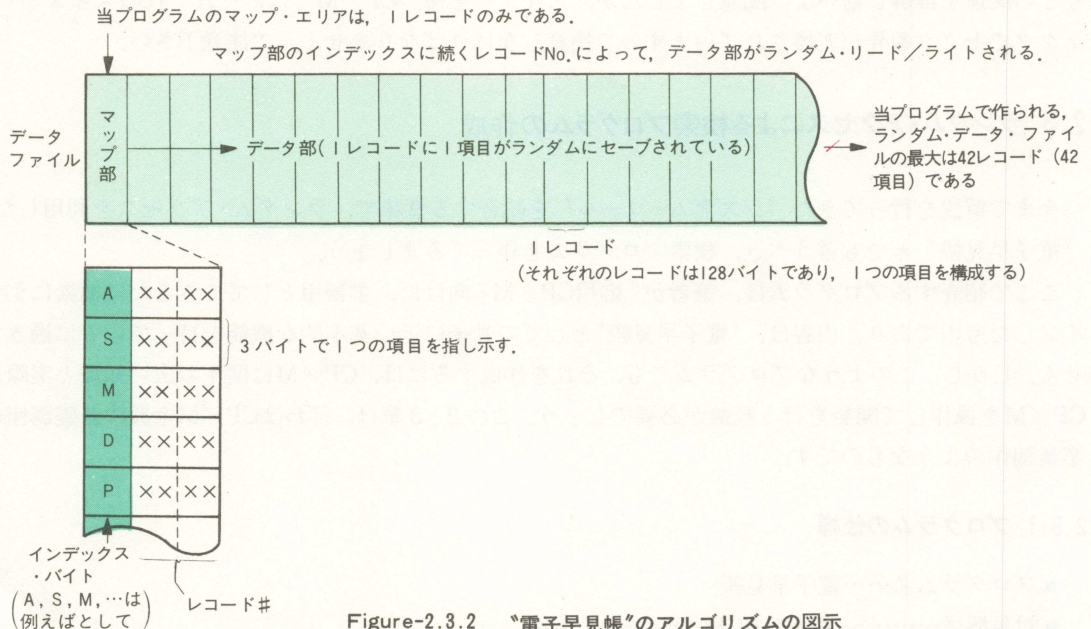


Figure-2.3.2 “電子早見帳”のアルゴリズムの図示

マップの初期設定

マップ部はファイルの先頭にあり、3バイト単位で1つの項目を指示しています。3バイトの内の最初のバイトは、インデックスとなるキーワード1文字が格納され、続く2バイトにはそのデータが格納される（セクタの）ランダム・レコードNoが格納されます。

マップ部は、新しくデータ・ファイルを作成する場合に、事前に必ず行わなければならない“初期設定”（Nコマンド）で、あらかじめ次の図に示すように各項目のレコードNoが書き込まれます。

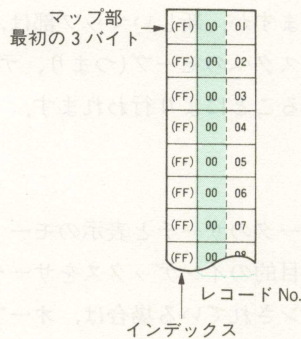


Figure-2.3.3 初期設定後のマップ部

このように、3バイト単位の各項目のインデックス部には、すべてにFFHが書き込まれ、それに対応するレコードNo部にはあらかじめ、0001, 0002, 0003……と、レコードNoが順番に書き込まれます。当プログラムでは、マップ部がわずか1レコードなので次のレコード、つまりデータ部の最初のランダム・レコードNoは、0001となります。もし、マップ部がnレコードを使用するならば、データ部の最初のランダム・レコードNoは、 $n + 1$ となります（このレコードNoの算出は、ファンクション：36で行います）。

インデックス部にFFHが書かれていると、その項目以後のデータは、“未記入”とみなされます。

このような初期状態のマップ部が、ファイルとしてディスク上に作られ、“初期設定”の作業が終わると、以後のディスク・アクセスは、このマップに従ってランダムにアクセスされ、データの書き込み、読み出しが行われます。

データの書き込み

データの書き込みモード（Iコマンド）にすると、該当ファイルがオープンされ、そのマップ部がデフォルトのDMAバッファ（80H～FFH）に読み込まれます。

プログラムは次に、メモリ上に読み込まれたマップ部の各インデックスをサーチし、FFHのインデックスを見つけます。そのレコードNoが、次に書き込みを行う項目に当たります。

次は、項目の頭文字を1文字キー入力します。入力された1文字は、マップ部の先程のインデックスのFFHであった部分に書き込まれます。

項目の頭文字を、メモリ上のマップ部のインデックス・ポイントに書き込むと、次はデータの入力です。

各項目のデータの文字数は、改行なども含めて127文字以下でなくてはなりません。キー入力されたデータは、当プログラムの最終部にある2nd DMAバッファに格納されて行きます。

データの入力の終わりを示すCtrl-Zを入力すると、2nd DMAバッファの内容が、先程、頭文字を書き込んだインデックスに続く2バイトが示すレコードNo.に書き込まれます。データ部はこのように、入力が終わると同時にディスクに書き込まれますが、新しいマップ部は、この時点ではまだディスクには書き込まれていません。マップ部のディスクへのセーブ(つまり、ディスク上のマップ部の更新)は、終了コマンドであるEコマンドを実行することにより行われます。

データの表示

Sコマンドをキーインすることにより、データのサーチと表示のモードになります。ファイルをオープンし、マップ部をメモリ上に読み出して目的のインデックスをサーチする動作は、データの入力の場合と同じです。ファイルがすでにオープンされている場合は、オープンの動作はスキップされます。

サーチしたい項目のインデックス(1文字)をキー入力すると、前記と同様に、メモリ上のマップ部をサーチして発見したインデックスに対するレコードNo.を取り出します。そして、そのレコードをランダム・リードし、その内容をコンソールに表示します。

プログラムの終了

Eコマンドのキーインにより、当プログラムを終了しCP/Mに戻ります。もし、マップ部の変更があった場合は、ディスク上の古いマップ部を新しいマップ部に書き替えます。変更がなかった場合は、何も行わずにCP/Mに戻ります。

注意事項

当プログラムの目的は、システム・コールの総合的な応用として、ランダム・ファイルの簡単な応用例を実習することにあります。よって、プログラムが複雑になることを避けるため、基本的な機能しか持たせていませんので、当プログラムの実行には、次の点に注意して下さい。

- ディスク上にすでに存在するファイル名で、新ファイルの作成コマンド(Nコマンド)を実行すると、同時に同名ファイルが作られてしまいます。

- ミスタイプを訂正することはできません。一発勝負です。
- すでに入力されているデータの削除・変更などはできません。
- 同じインデックスを入力した場合は、最初に入力したものが読み出されます。
- 項目数は、たったの42項目しか収容できません。項目数の制限を超えても警告は出ません。暴走することになるでしょう。また、一項目の最大の文字数127文字についても同様です。ただし、これについては127文字を超えても暴走することはありません。

もちろん、これらの点をクリアすることや、使い勝手を良くすることは、実用プログラムとしては不可欠のことですが、ここでは基本的な構成が、それらによって見えなくなってしまう意味がありませんので我慢して下さい。

次に当“電子早見帳”プログラムのフローチャートを示します。

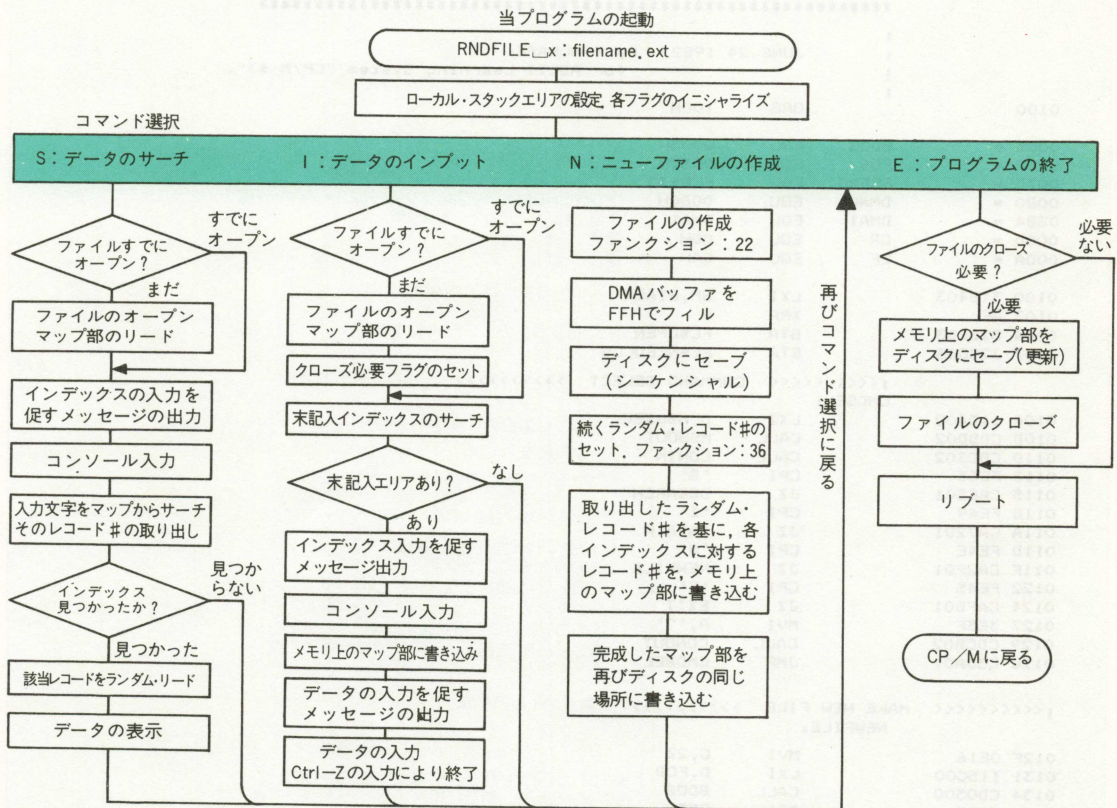


Figure-2.3.4 “電子早見帳”のフローチャート

す。このリストを基に、
"RNDFILE.COM"

ントを見ながら追って行

ランダム・アクセスによる検索プログラムの作成

013C 218000	LXI	H, OOB0H	
013F 1EFF	MVI	E, OFFH	
0141 0680	MVI	B, 12B	
0143 73	MAPWT:		
0144 23	MOV	M, E	
0145 05	INX	H	
0146 C24301	DCR	B	
	JNZ	MAPWT	
0149 AF	XRA	A	
014A 327C00	STA	FCB+32	
014D CD9602	CALL	SEQWRITE	
0150 CDB702	CALL	SETRNDRCO	
0153 EB	XCHG		
0154 218000	LXI	H, DMA0	
0157 062A	MVI	B, 12B/3	
0159 23	INX	H	
015A 73	MOV	M, E	
015B 23	INX	H	
015C 72	MOV	M, D	
015D 23	INX	H	
015E 13	INX	D	
015F 05	DCR	B	
0160 C25901	JNZ	INSLMAP	
0163 361A	MVI	M, 1AH	
0165 AF	XRA	A	
0166 327C00	STA	FCB+32	
0169 CD9602	CALL	SEQWRITE	
016C CD7902	CALL	CLOSE	
016F C30A01	JMP	CMDSEL	
0172 3AB0C3	LDA	FL*OPEN	
0175 3C	INR	A	
0176 C8B101	JZ	DATIN1	
0179 CD1A02	CALL	DPNMPRD	
017C 3EFF	MVI	A, OFFH	
017E 32BD03	STA	FL*NEEDCLS	
0181 3EFF	MVI	A, OFFH	
0183 CD4C02	CALL	SEARCH	
0186 3C	INR	A	
0187 CA0A01	JZ	CMDSEL	
018A 112A03	LXI	D, MSGIDXIN	
018D CDBD02	CALL	MSGOUT	
0190 CDC302	CALL	CONIN	
0193 2ABE03	LHLD	PT*IDAD	
0196 77	MOV	M, A	
0197 115203	LXI	D, MSGDATIN	
019A CDBD02	CALL	MSGOUT	
019D 21B403	LXI	H, DMA1	
01A0 CDC302	CALL	CONIN	
01A3 77	MOV	M, A	
01A4 23	INX	H	
01A5 FE1A	CPI	1AH	
01A7 C8B901	JZ	RNDWT	
01AA FE0D	CPI	CR	
01AC C2A001	JNZ	KEYIN	
01AF 3E0A	MVI	A, LF	
01B1 77	MOV	M, A	
01B2 CDCB02	CALL	CONOUT	
01B5 23	INX	H	
01B6 C3A001	JMP	KEYIN	

0243 0E1A	MVI	C, 26	これらのDMA/バッファを、2ndDMA/バッファのDMA1に設定。
0245 11B403	LXI	D, DMA1	
0248 CD0500	CALL	BDOS	
024B C9	RET		

```

; <<<<<<<<<  SEARCH INDEX LETTER  >>>>>>>>>
SEARCH:                                     (INDEXのサーチ)

```

```

024C 47          MOV      B,A
024D 218000       LXI      H,DMAO

```

NXTIDX: 66530

```

0250 7E          MOV     A,M
0251 FE1A       CPI     1AH
0253 CA6D02    JZ      NOTFIND
0256 BB       CMP     B
0257 CA6002    JZ      JUST
025A 232323    INX H! INX H! INX H
025D C35002    JMP     NXTIDX

```

Aレジスタに格納されている、サーチの対象文字を、DMA0バッファ内の各INDEX部から探し出す。見つければ、そのINDEXのDMA 0 内のアドレスを“PT-\$IDAD”にセット、それに対応するランダム・レコードNo.を、“PT-\$IDAD”にセット、そのINDEXが見つからない場合は、AレジスタにFFHをセットしてリターン。

見つかった場合は、00をセットしてリターン。

PC	OPCODE	JUST:	OP	DATA
0260	228E03		SHLD	PT*IDAD
0263	23		INX	H
0264	5E		MOV	E,M
0265	23		INX	H
0266	56		MOV	D,M
0267	EB		XCHG	
0268	229203		SHLD	PM*RECC
026B	AF		XRA	A
026C	C9		RFT	

```
NOTFIND:      MVI      A, OFFH
```

```
026D 3EFF          MVI      A, OFFH
026F C9           RET
```

```
; <<<<<<<<<< ERROR >>>>>>>>> (すべてのエラー処理)
ERROR:
```

```

0270 111E03      LXI      D,MSGEN
0273 CDBD02      CALL     MSGOUT
0276 C30000      JMP      0000H

```

エラーメッセージを出力し、リブートしてCP/Mに戻る

```
; <<<<<<<< FILE CLOSE >>>>>>>> (ファイルのクローズ)
CLOSE;
```

```
0279 0E10          MVI      C,16
027B 115C00        LXI      D,FCB
027E CD0500        CALL    BDOS
0281 FEFF          CPI      255
0283 CA7002        JZ       ERROR
0286 C9            RET
```

ファイルのクローズ

```
; <<<<<<<<< SET RANDOM RECORD >>>>>>>>>
SETRNDRCO: (ランダム・レコードNo.のセット)
```

0287 0E24	MVI	C, 36
0289 115C00	LXI	D, FCB
028C CD0500	CALL	BDOS
028F 2A7D00	LHLD	FCB+33
0292 229003	SHLD	PM\$RECO
0295 C9	RET	

シーケンシャルなリード／ライトに続く、ランダム・レコードNo. をFCBのr0, r1フィールドにセットする。

```
; <<<<<<<<<< SEQUENTIAL WRITE >>>>>>>>> (シーケンシャルな書き込み)
```

```

0296 0E15          MVI      C,21
0298 115C00        LXI      D,FCB
029B CD0500        CALL     BDOS
029E B7           ORA      A
029F C27002        JNZ      ERROR
02A2 C9           RET

```

シーケンシャルな書き込み。

```

; <<<<<<<<<<  RANDOM READ 1 RECORD  >>>>>>>>> (ランダム・リード)
RNDREAD:

```

```

02A3 0E21          MVI     C,33
02A5 115C00        LXI     D,FCB
02A8 CD0500        CALL   BDOS
02AB B7            ORA     A
02AC C27002        JNZ     ERROR
02AF C9            RET

```

ランダムなリード。


```

; <<<<<<<<<< RANDOM WRITE 1 RECORD >>>>>>>>> (ランダム・ライト)
RNDWRITE:
02B0 0E22          MVI      C,34
02B2 115C00        LXI      D,FCB
02B5 CD0500        CALL     BDOS
02B8 B7            ORA      A
02B9 C27002        JNZ      ERROR
02BC C9            RET

; <<<<<< MESSAGE OUT. STRING ADR ON REG.D,E >>>>> (メッセージ・アウト)
MSGOUT:
02BD 0E09          MVI      C,9
02BF CD0500        CALL     BDOS
02C2 C9            RET

; <<<<<<<< CONSOLE INPUT. DATA ON REG.A >>>>>>>> (コンソール入力)
CONIN:
02C3 E5            PUSH     H
02C4 0E01          MVI      C,1
02C6 CD0500        CALL     BDOS
02C9 E1            POP      H
02CA C9            RET

; <<<<<<<<< CONSOLE OUT. REG.A OUT >>>>>>>> (コンソール出力)
CONOUT:
02CB E5            PUSH     H
02CC 0E02          MVI      C,2
02CE 5F            MOV      E,A
02CF CD0500        CALL     BDOS
02D2 E1            POP      H
02D3 C9            RET

; 以下メッセージ文字列エリア
02D4 0D0A0D0A53 MSGSEL: DB CR,LF,CR,LF,'SELECT COMMAND',CR,LF
02E8 5329656172 DB 'Search, Input, New file, Exit, ( S/I/N/E )? >*'
031E 0D0A455252 MSGERR: DB CR,LF,'ERROR !',CR,LF,'*'
032A 0D0A0D0A49 MSGIDXIN: DB CR,LF,CR,LF,'INPUT INDEX LETTER (1 CHARACTER) >*'
0352 0D0A0D0A49 MSGDATIN: DB CR,LF,CR,LF,'INPUT DATA',CR,LF,CR,LF,'>*'
0366 0D0A0D0A49 MSGSRCH: DB CR,LF,CR,LF,'INPUT SEARCH INDEX LETTER >*'
03B7 0D0A0D0A24 MSGCRLF: DB CR,LF,CR,LF,'*'

038C              FL*OPEN: DS 1
038D              FL*NEEDCLS: DS 1
038E              PT*IDAD: DS 2
0390              PM*RECO: DS 2
0392              PM*RECC: DS 2
0394              DS 32
03B4 -            STACK EQU *
03B4              DMA1: DS 128
0434              END

A>

```

ランダムな書き込み。

D, Eレジスタでアドレスされるメッセージを、コンソールに出力。

コンソールから1文字入力。

Aレジスタの文字を、コンソールに出力。

各フラグ、アドレス・ポインタ、パラメータ、DMA1/ハップファ、スタックの各エリア。

Figure-2.3.5 “電子早見帳” プログラムのソース・リスト。

2.3.3 電子早見帳プログラムの実行

当プログラムを実行するためのコマンド形式を次に示します。

RNDFILE x : filename . ext]

ここで、x : はドライブ名 (ログイン・ディスクの場合は省略できる)、filename . ext は、検索を行おうとするデータ・ファイル名、あるいは、新しく作成しようとするデータ・ファイル名のことです。

新データ・ファイルの作成

ではプログラムを実行してみましょう。まだデータ・ファイルは何も作られていませんので、まず、「新ファイルの作成」を行います。

入力するデータは、住所録とか、何かのリストとか、適当なものとしませんが、ここで示す例は、大抵の本の巻末にある「ページ索引」を利用しました。使った本は、「The CP/M handbook」です。

新しく作成する「ページ索引」のデータ・ファイル名を、「CPMINDEX.IDX」として実行した例を次に示します。

A>RNDFILE CPMINDEX.IDX)新しいファイル「CPMINDEX.IDX」を作り、データを書き込む。

SELECT COMMAND

S)earch, I)ntput, N)ew file, E)xit, (S/I/N/E)? >N新ファイルの作成コマンドを実行。

SELECT COMMAND

S)earch, I)ntput, N)ew file, E)xit, (S/I/N/E)? >Iデータの入力モードを選択。

INPUT INDEX LETTER (1 CHARACTER) >Aインデックス「A」の項目を入力。

INPUT DATA

```
>ABORT : 101,219
Active User : 76
Append : 131
ASM : 81,221
ATTACH : 102,212,223
^Z
```

→ 項目「A」のデータを入力。
各ラインの終わりに、CR/LFを入力すること。
LF(改行)はCtrl-Jで代用できる。
それぞれの項目のデータ容量は128バイト以内。
データの最後にはCtrl-Zを入力する。

SELECT COMMAND

S)earch, I)ntput, N)ew file, E)xit, (S/I/N/E)? >I入力モード選択。

INPUT INDEX LETTER (1 CHARACTER) >Sインデックス「S」を入力。

INPUT DATA

```
>SAVE : 30,249
SCHED : 25,94,100
SPOOL : 94,257
STAT : 67,68,259
SUBMIT : 76,263
System Disket : 1,13,23,65
^Z
```

→ 項目「S」のデータを入力。

SELECT COMMAND

S)earch, I)ntput, N)ew file, E)xit, (S/I/N/E)? >I入力モード選択。

INPUT INDEX LETTER (1 CHARACTER) >Mインデックス「M」を入力。

INPUT DATA

```
>Machine language : 81
MBASIC : 42
MDS-800 : 191
Memory : 2
MOVECPM : 198,240,243
MP/M : 209
^Z
```

→ 項目「M」のデータを入力。

SELECT COMMAND

S)earch, I)ntput, N)ew file, E)xit, (S/I/N/E)? >I入力モード選択。


```
INPUT INDEX LETTER (1 CHARACTER)
INPUT DATA
```

.....インデックス"D"を入力。

```
>DDT : 86,225
Debugging : 86
DELETE : 22
Directory : 23,61
Disk drive : 20
Double-density : 71
^Z
```

← 項目"D"のデータを入力。

```
SELECT COMMAND
```

```
S)earch, I)nter, N)ew file, E)xit, ( S/I/N/E )? >I .....入力モード選択。
```

```
INPUT INDEX LETTER (1 CHARACTER) >P .....インデックス"P"を入力。
INPUT DATA
```

```
>Parity : 136
Password : 187
PIP : 34,40,109,249
Printer : 3
Process : 105
PUTSYS : 198
^Z
```

← 項目"P"のデータを入力。

```
SELECT COMMAND
```

```
S)earch, I)nter, N)ew file, E)xit, ( S/I/N/E )? >I .....入力モード選択。
```

```
INPUT INDEX LETTER (1 CHARACTER) >B
INPUT DATA
```

```
>Backup : 37
BASIC : 26
BDOS : 68,182,185,194
Binary : 81
Boot : 13,194
Bootstrap loader : 5
Byte : 68
^Z
```

← 項目"B"のデータを入力。

.....

```
SELECT COMMAND
```

```
S)earch, I)nter, N)ew file, E)xit, ( S/I/N/E )? >E .....ディスク上のマップ部が更新され、
プログラムを終了。
```

```
A>
```

Figure-2.3.6 “電子早見帳”の実行。新データ・ファイルを作成する。

データの検索

いよいよ電子早見帳の利用です。先程作成した“ページ索引”のデータ・ファイルを検索してみましょう。

実行例を次に示します。

A>RNDFILE CPMINDEX.IDX /すでにデータが書き込まれているファイル"CPMINDEX.IDX"に対して、当プログラムを実行。

SELECT COMMAND

S)earch, I)ntput, N)ew file, E)xit, (S/I/N/E)? >Sデータのサーチを行う。

INPUT SEARCH INDEX LETTER >Dサーチ・インデックスとして、"D"を入力。

DDT : 86,225
Debugging : 86
DELETE : 22
Directory : 23,61
Disk drive : 20
Double-density : 71

← "D"に関する項目が読み出された。

SELECT COMMAND

S)earch, I)ntput, N)ew file, E)xit, (S/I/N/E)? >S再びデータのサーチ。

INPUT SEARCH INDEX LETTER >Pインデックス"P"をサーチ。

Parity : 136
Password : 187
PIP : 34,40,109,249
Printer : 3
Process : 105
PUTSYS : 198

← "P"に関する項目が読み出された。

SELECT COMMAND

S)earch, I)ntput, N)ew file, E)xit, (S/I/N/E)? >S再びデータのサーチ。

INPUT SEARCH INDEX LETTER >Mインデックス"M"をサーチ。

Machine language : 81
MBASIC : 42
MDS-800 : 191
Memory : 2
MOVECPM : 198,240,243
MP/M : 209

← "M"に関する項目が読み出された。

SELECT COMMAND

S)earch, I)ntput, N)ew file, E)xit, (S/I/N/E)? >S再びデータのサーチ。

INPUT SEARCH INDEX LETTER >Aインデックス"A"をサーチ。

ABORT : 101,219
Active User : 76
Append : 131
ASM : 81,221
ATTACH : 102,212,223

← "A"に関する項目が読み出された。

.....

SELECT COMMAND

S)earch, I)ntput, N)ew file, E)xit, (S/I/N/E)? >Eプログラムを終了。

A>

Figure-2.3.7 "電子早見帳"の実行、データ・ファイルの検索。

2.3.4 ディスク内部におけるデータの記録状態

ディスク上にファイルされたデータが、実際にどのように記録されているかを見てみましょう。今作成した“CPMINDEX . IDX”のデータ・ファイル（6項目のみ入力されている）のマップ部とデータ部の最初である、“A”の項のレコードをセクタ・ダンプして示します。どここのセクタに格納されているかは、ディスクの他のファイルの格納状況によりますので、データ・ファイルのマップ部からでは探し出すことはできません。

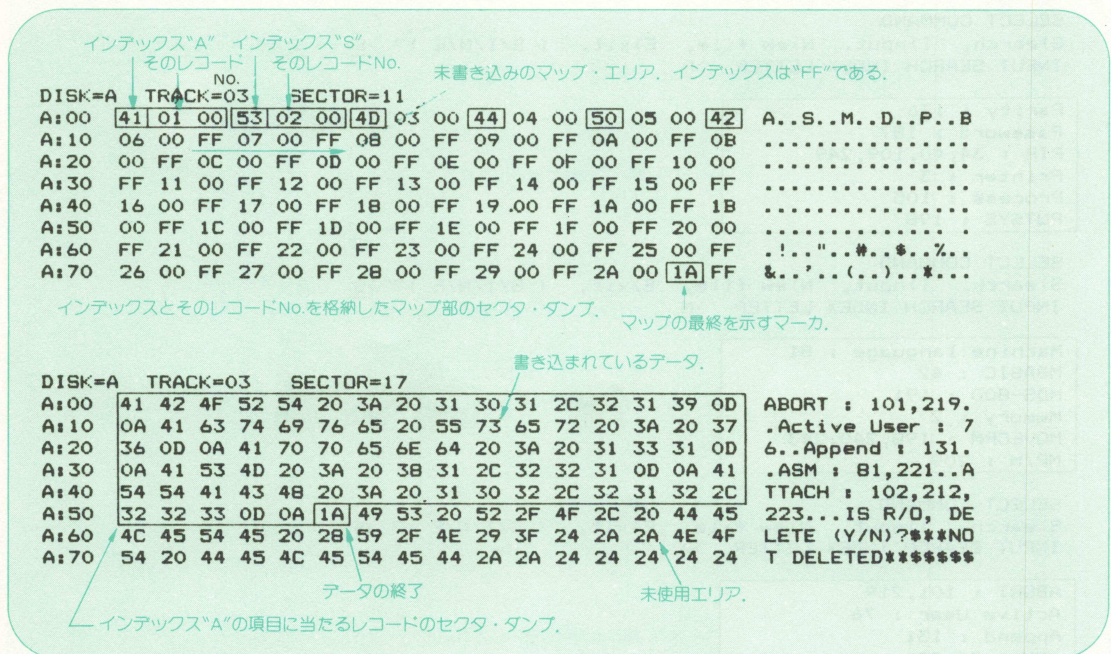
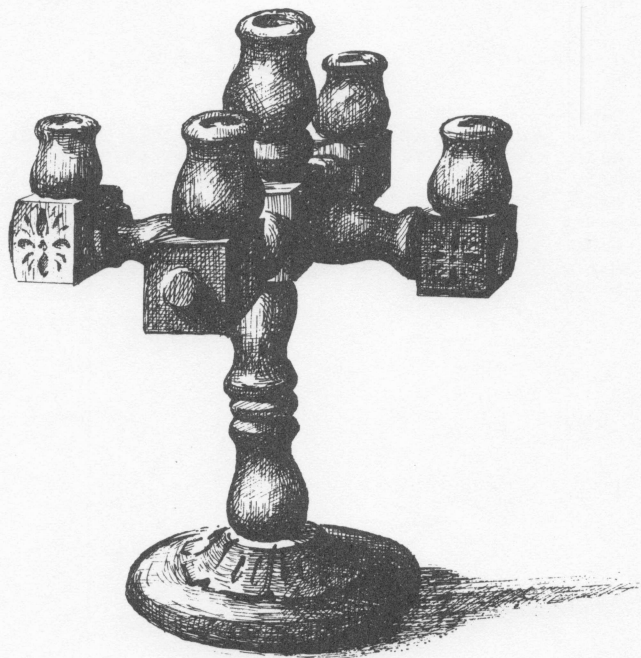


Figure-2.3.8 データ・ファイルのディスク上の記録状態。

3章 マクロ・アセンブラおよび リンク・ローダによるソフト開発



ここでは、代表的なアセンブラである“MAC”，“RMAC”，“MACRO-80”の3種類と，リンク・ローダの“LINK-80”，それに，シンボリック・インストラクション・デバッガの“SID”，“ZSID”を取り上げ，それぞれの機能の概要や，実際のソフトウェア開発（8080およびZ80）などを紹介します。

本書で，それぞれを詳細に語ることは不可能（それぞれが厚いマニュアルとなる）なので，ここで紹介するのは機能の一部，使い方の一例であることをお断りしておきます。

3.1 MACと(Z)SIDの使用例とマクロ・ライブラリの利用法

デジタルリサーチ社のマクロ・アセンブラである“MAC”は，CP/Mユーザーの中では，標準アセンブラの“ASM”以外で，恐らく一番多く使われているアセンブラではないかと思います。

例えば，NECのPC-8000シリーズCP/Mも，PDA-800 CP/Mも，沖のif-800 CP/Mも，富士通のFM-8 CP/Mも，この“MAC”を使って作られています。他の機種種のCP/Mも，ほとんどはこのアセンブラを使って作られていることでしょう。

このアセンブラには，後程解説しますが，CP/Mのシステム・コールを利用した各種サブルーチンが，マクロ・ライブラリとして提供されています。また，1章で解説した，CP/Mを各機種にインプリメントする場合に便利なライブラリも提供されています。よって，特にCP/Mをインプリメントしたり，CP/M上で実行する各種ソフトを開発するユーザーにとって，一度は使わなければならないであろう必需品と言えるでしょう。

3.1.1 MACの機能

MACは，インテル形式の8080ニーモニック，およびインテル・ニーモニック拡張形式のZ80ニーモニックをアセンブルし，インテルHEX形式のオブジェクト・ファイルと，PRNリスト・ファイル，それにシンボル・テーブルのリスト・ファイルを出力する，マクロ機能を持ったアセンブラです。ただし，リロケート可能なオブジェクト・コードを出力する機能はありません。

Z80のアセンブルは，アセンブラ本体にZ80のアセンブラが内蔵されている訳ではなく，“Z80.LIB”というZ80をアセンブルするためのライブラリを，マクロ機能を使って取り込むことにより実現しています。同様に“I8085.LIB”により，8085にも対応できます。

3.1.2 MACの使い方実例

マクロ定義と定義されたマクロの使用例

最初に，代表的な使い方の1つとして，ユーザー独自の2～3のマクロを定義して，それを同一プログラム中で使用する簡単な例を示します。

定義するマクロは、次の3種です。

マクロ名

機能

ALLPUSH すべてのレジスタの退避を行う。

ALLPOP ALLPUSHと逆に、すべてのレジスタの復帰を行う。

MSGOUT ダミー・パラメータ部に記述したメッセージをコンソールに出力する。そして、メッセージの前後には、自動的に復帰・改行を挿入する。

上記の3種のマクロを定義して、これらを利用するプログラムを1つのソース・プログラムとして作成します。このソース・プログラム例を次に示します。ファイル名のエクステンションは、“ASM”でなければなりません。

```

A>TYPE SMPL1MAC.ASM) .....マクロ定義とそれを使用したサンプル・プログラムのソース・ファイル。
                           ファイル名のエクステンションは、“ASM”でなければならない。
*****
**          MAC SAMPLE PROGRAM 1          **
**          STORED MACRO                  **
*****
      ORG      100H                      ユーザー独自のマクロ名を定義。
; <<<<<<<<<< MACRO DEFINITION >>>>>>>>
[ALLPUSH] [MACRO] .....“ALLPUSH”というマクロ名で、MACRO~ENDM間に記述されたステートメント
      PUSH     PSW      を登録する。
      PUSH     B         この部分が、マクロ名“ALLPUSH”として登録される。
      PUSH     D         (すべてのレジスタの退避)
      PUSH     H
      [ENDM]

[ALLPOP] [MACRO]
      POP      H
      POP      D         上と同様に、この部分がマクロ名“ALLPOP”として登録される。
      POP      B         (すべてのレジスタの復帰)
      POP      PSW
      [ENDM]

[MSGOUT] [MACRO] .....ダミー・パラメータ
      LOCAL    MESS
      MESBUF, MOUT
      JMP      MOUT
      MESBUF: DB    ODH, 0AH, '&MESS', ODH, 0AH, '*'
      MOUT:    MVI    C, 9
      LXI      D, MESBUF
      CALL     0005H
      [ENDM]
.....同様に、マクロ名“MSGOUT”として登録する。(ダミー・パラメータ“MESS”の部分に置かれる文字列をコンソールに表示する)

;
; <<<<<<<<<< MAIN PROGRAM >>>>>>>>
      ALLPUSH
      MSGOUT    <*** THIS IS MAC SAMPLE PROGRAM 2 ***> .....実際の文字列
      ALLPOP
      MSGOUT    <END> .....実際の文字列
      RET
      END

```

上で定義したマクロ名を利用して、簡単なプログラムを作る。

すべてのレジスタを退避して、メッセージを表示し、すべてのレジスタを復帰する。そして別のメッセージを表示する。というプログラムである。

Figure-3.1.1 マクロ定義と定義されたマクロの使用例のサンプル・プログラムのソース・ファイル。

次にソース・ファイルを、MACを使ってアセンブルします。アセンブルを実行する時のコマンド・ラインに、アセンブリ・パラメータを付けることにより、アセンブル実行時の入力／出力ファイルの各種のコントロールなどが行えますが、ここでは一番単純なコマンドで実行します。

アセンブルの実行例を次に示します。

```
A>MAC SMPL1MAC / .....ソース・ファイル"SMPL1 MAC. ASM"をMACでアセンブルする。
CP/M MACROD ASSEM 2.0
0150
002H USE FACTOR
END OF ASSEMBLY
A> .....アセンブル終了。
```

Figure-3.1.2 MACによるアセンブルの実行。

アセンブルが正常に終了しました。ソース・ファイル "SMPL1MAC. ASM" から、次の各ファイルが生成されました。DIRで示します。その後で、LOADコマンドにより "COM" ファイルを生成し、再びDIRで各ファイルを確認してみましょう。

```
A>DIR SMPL1MAC.* / .....アセンブルにより作られた各種ファイルの確認。
A: SMPL1MAC ASM : SMPL1MAC PRN : SMPL1MAC HEX : SMPL1MAC SYM
   ソース・ファイル   リスト・ファイル   HEXオブジェクト・ファイル ↑
A>LOAD SMPL1MAC / .....LOADコマンドで
                        COMファイルに変換。
FIRST ADDRESS 0100
LAST ADDRESS 014F
BYTES READ 0050
RECORDS WRITTEN 01
MACでは、シンボルテーブルのファイルも
作られる。ただしこの例ではメインプログラ
ムにシンボルを使ってないので、ファイ
ルの内容は空である。(後述)
A>DIR SMPL1MAC.* / .....生成されたCOMファイルの確認。
A: SMPL1MAC COM : SMPL1MAC ASM : SMPL1MAC PRN : SMPL1MAC HEX
A: SMPL1MAC SYM
A>
```

Figure-3.1.3 MACによって生成された各ファイルの確認と、LOADコマンドによる "COM" ファイルの生成。

以上の手順で、実行可能な "COM" ファイル "SMPL1MAC. COM" ができ上がりました。とりあえず、このプログラムを実行してみましょう。

```
A>SMPL1MAC / .....サンプル・プログラムの実行。
*** THIS IS MAC SAMPLE PROGRAM 2 ***
END
A>
```

表示されたこの2つのメッセージと、Figure-3.1.1のメインプログラム部を比較して下さい。

Figure-3.1.4 サンプル・プログラムの実行。

このように、Figure-3.1.1のソース・ファイルのメインプログラム部に書かれた2種類のメッセージ（<>で囲まれた文字列）が、コンソールに表示されています。

さて次に、本項の解説で、最も重要なリストとなる当サンプル・プログラムのPRNファイルを示します。“マクロ”の概念を、このPRNリストから理解することができるでしょう。

ソース・ファイルであるFigure-3.1.1のリストと、次のPRNファイルのリストとをよく対比してみてください。定義されたそれぞれのマクロが、メインプログラムで使用され、それがアセンブルされることにより、どのように展開され、オブジェクト・コードに変換されるかに注目して下さい。

A>TYPE SMPL1MAC.PRN /-----PRNファイルにより展開されたそれぞれのマクロ定義を見ることができる。

```

*****
**                               MAC SAMPLE PROGRAM 1                               **
**                               STORED MACRO                                       **
*****

0100          ORG      100H

; <<<<<<<<<<  MACRO DEFINITION  >>>>>>>>

ALLPUSH MACRO
    PUSH    PSW
    PUSH    B
    PUSH    D
    PUSH    H
    ENDM

ALLPOP  MACRO
    POP     H
    POP     D
    POP     B
    POP     PSW
    ENDM

MSGOUT  MACRO    MESS
    LOCAL    MESBUF, MOUT
    JMP      MOUT
MESBUF:  DB      0DH, 0AH, '&MESS', 0DH, 0AH, '*'
MOUT:    MVI     C, 9
          LXI     D, MESBUF
          CALL    0005H
          ENDM

; <<<<<<<<<  MAIN PROGRAM  >>>>>>>>

ALLPUSH
    PUSH    PSW
    PUSH    B
    PUSH    D
    PUSH    H

MSGOUT <*** THIS IS MAC SAMPLE PROGRAM 2 ***>
    JMP     ??0002
0107+0D0A2A2A2A??0001: DB      0DH, 0AH, '*** THIS IS MAC SAMPLE PROGRAM 2 ***', 0DH, 0AH, '*'
0130+0E09          ??0002: MVI     C, 9
0132+110701          LXI     D, ??0001
0135+CD0500          CALL    0005H

ALLPOP
    POP     H
    POP     D
    POP     B
    POP     PSW

```

マクロ定義部は、ソース・ファイルと同じ。

“+”記号は、マクロによって展開されたコードであることを示す。

展開されたメインプログラム部。ソース・ファイルとよく比較して下さい。


```

013C+C34701
013F+0D0A454E44??0003: DB      0DH,0AH,'END',0DH,0AH,'$'
0147+0E09      ??0004: MVI      C,9
0149+113F01      LXI      D,??0003
014C+CD0500      CALL     0005H
014F C9      RET
0150      END

```

←ここはマクロではないので"+"記号がない。

A>

Figure-3.1.5 サンプル・プログラムのPRNファイルのリスト、マクロの概念を知る上で最重要。

このサンプル・プログラムは、マクロ定義とその使用（マクロ・コールと言う）の最も単純な例であり、本来は、もっと高度な定義法や使用方法があります。

いずれにしても、よく使われるようなルーチンをマクロで定義しておけば、使う時に、マクロ名、あるいはマクロ名と必要なパラメータを記述するだけで良いのです。あとはMACアセンブラが自動的にFigure-3.1.5のように展開してくれます。パラメータを変えて、同じルーチンと呼ぶことができる点が、普通のサブルーチンと大きく異なります。

マクロ・ライブラリの利用とその使用例

購入したMACのディスクettには、本体の“MAC.COM”の他に、多くのマクロ・ライブラリが含まれています。その内容を記したファイル、“DISK.DOC”をタイプアウトして次に示しますので参考にして下さい。

A>TYPE DISK.DOC!.....“MAC”のディスクettに含まれる各種ファイルについての説明文のタイプアウト。

File:	Contents:
MAC.COM	"MAC" Macro Assembler
SAMPLE.ASM	Sample program to test MAC execution
IB085.LIB	Simple macros for 8085 RIM/SIM instructions
Z80.LIB	Macro library for Z80 operation codes
Z80.DOC	Documentation for the Z80.LIB file
INTER.LIB	Traffic light intersection library (see manual)
TREADLES.LIB	Library for traffic treadles
BUTTONS.LIB	Library for pedestrian pushbuttons
SIMP10.LIB	Simple BDOS I/O Library
SEQ10.LIB	Sequential file I/O library
STACK.LIB	Simple stack machine library
DSTACK.LIB	Complete stack machine library
COMPARE.LIB	Library for simple 8-bit comparison operations
NCOMPARE.LIB	8-bit comparisons with negation
WHEN.LIB	Macros for the WHEN construct (see manual)

DOWHILE.LIB
SELECT.LIB

Macros for the DOWHILE construct
Macros for the SELECT construct

A>

Figure-3.1.6 “MAC” のディスクットに含まれる各種ファイルの説明文のタイプアウト。

これらのライブラリを利用することにより、8080, 8085, Z80のいずれをもアセンブルすることが可能となりますが、まず、これらのマクロ・ライブラリの中から、シーケンシャル・ファイル I/O ライブラリ、“SEQIO.LIB”を利用して、サンプル・プログラムを作ってみましょう。

CP/Mのビルトイン・コマンドの“TYPE”と同等のプログラムを作ってみましょう。もし、マクロ・ライブラリを利用せずに、このプログラムを実現するには、2.2章のシステム・コールのファンクション：15, 20項で実習したプログラムを骨子とすれば良いでしょう。

しかし、このプログラムにマクロ・ライブラリを利用した場合、どのような威力を発揮するか、まずその実例を示します。

“SEQIO.LIB”マクロ・ライブラリを利用した、アスキー・ファイルのタイプアウト・プログラム（プログラム名“TYPEOUT”）のソース・リストを示します。

A>TYPE TYPEOUT.ASM!「マクロ・ライブラリの利用」サンプル・プログラムのソース・ファイルをタイプアウト。

```

*****
**          MAC SAMPLE PROGRAM 2          **
**          ASCII FILE TYPEOUT            **
*****

ORG      100H

MACLIB  SEQIO .....マクロ・ライブラリ“SEQIO.LIB”ファイルの使用を宣言。

LXI     SP,STACK .....ローカル・スタックポインタ設定。

FILE    INFILE,INPUTFILE,,1,1024 .....“FILE”マクロの使用。1st FCBにあるファイルをオープン
NXTBLOCK:                                     し、1024バイトのバッファに読み出せるように準備する。
[GET    INPUTFILE] .....“GET”マクロの使用。“INPUTFILE”(1st FCBにあるファイル)を読み出す。
JZ      0000H
CPI     1AH .....エンドオブファイルであればリポートしてCP/Mに戻る。
JZ      0000H
[PUT    CON] .....“PUT”マクロの使用。“CON”(コンソール)に出力する。
JMP     NXTBLOCK

DS      32
STACK   EQU    $
BUFFERS: .....これ以後は、1024バイトのファイル読み出し時のバッファに使用。
END

A>
```

Figure-3.1.7 マクロ・ライブラリを利用した、タイプアウト・プログラムのソース・リスト。

このソース・リストのようにSEQIOマクロ・ライブラリの中から、“FILE”、“GET”、“PUT”の3つをマクロ・コールすることにより、実に簡素にタイプアウト・プログラムを実現することができます。

ここで、“FILE”マクロのパラメータについて説明しておきましょう。

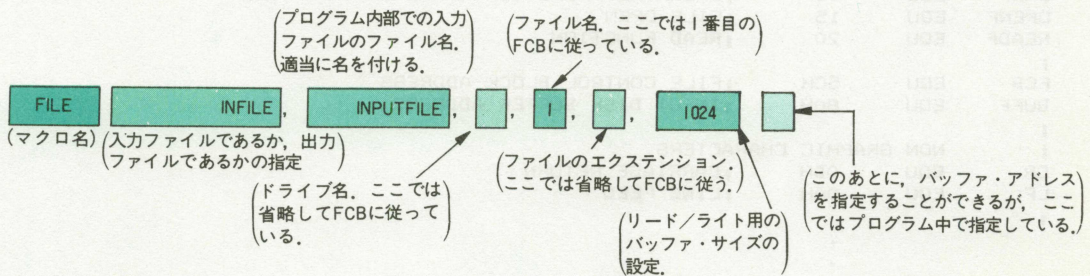


Figure-3.1.8 “FILE”マクロのパラメータ

当プログラムで用いた“FILE”マクロのパラメータは、このように入力ファイルとしての設定をしましたが、例えばもう1組、アウトプット・ファイル用の“FILE”のマクロ・コールを行い、プログラム中の“PUT CON”を、“PUT OUTPUTFILE”とでもすれば、このプログラムは、PIPコマンドのファイル・コピーのプログラムに相当するものになる訳です。

次に、このソース・ファイルをMACでアセンブルし、LOADコマンドを実行して、“TYPEOUT.COM”を生成し、実行してみましょう。アセンブラの実行は、Figure-3.1.2と同様なのでここで示すのは省略します。

ライブラリ・ファイルの“SEQIO.LIB”は、アセンブラが実行されるドライブと同一のドライブ上になければなりませんのでアセンブル時には注意して下さい。

でき上がったプログラムの実行時のコマンドは、

```
TYPEOUT x: filename. ext ]
```

です。x:はドライブ名で、ログイン・ディスクの場合は省略できます。

次に実行例として、ドライブB:上のファイル“DUMP.ASM”をタイプアウトしたものを示します。

```
A>TYPEOUT B:DUMP.ASM ] -----ドライブB:上のファイル、“DUMP.ASM”をタイプアウトする。

;      FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
;      COPYRIGHT (C) 1975, 1976, 1977, 1978
;      DIGITAL RESEARCH
;      BOX 579, PACIFIC GROVE
;      CALIFORNIA, 93950
;
```



```

      ORG      100H
BDOS   EQU     0005H      ;DOS ENTRY POINT
CONS   EQU     1         ;READ CONSOLE
TYPEF   EQU     2         ;TYPE FUNCTION
PRINTF   EQU     9         ;BUFFER PRINT ENTRY
BRKF    EQU     11        ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
OPENF   EQU     15        ;FILE OPEN
READF   EQU     20        ;READ FUNCTION
;
FCB     EQU     5CH       ;FILE CONTROL BLOCK ADDRESS
BUFF    EQU     80H       ;INPUT DISK BUFFER ADDRESS
;
;      NON GRAPHIC CHARACTERS
CR       EQU     0DH       ;CARRIAGE RETURN
LF       EQU     0AH       ;LINE FEED
;
      .
      .
      .

```

ファイルのEOFを検出した時点でファイルの出力を終わる。

Figure-3.1.8 マクロ・ライブラリを利用して作られたタイプアウト・プログラムの実行。

次に当プログラムで利用した "FILE" マクロが、ファイル操作上の各種エラー処理を行うルーチンを含んでいることを実際に確認してみましょう。

A>TYPEOUT ABCD.XYZ)ディスク上に存在しないファイルのタイプアウトを行う。

NO INPUTFILE FILEエラー・メッセージが表示されてCP/Mに戻った。

A> \ プログラム中で、内部のファイル名として指定した名前であることに注目。

Figure-3.1.9 "FILE" マクロのエラー処理の確認。ファイルが見つからない場合。

この他にも、"DISK FULL"とか、"CANNOT CLOSE"とか、各種のエラー処理を勝手に行ってくれます。

次に、当プログラムの中でマクロ・コールした部分は、実際にはどのように展開されているのか、"PRN" ファイルをタイプアウトして見てみましょう。

A>TYPE TYPEOUT.PRN ! -----アセンブルにより生成されたPRNファイルのタイプアウト.

```

*****
**          MAC SAMPLE PROGRAM 2          **
**          ASCII FILE TYPEOUT           **
*****

0100                ORG      100H

                        MACLIB  SEQIO

0100 312502        LXI      SP,STACK

                        FILE    INFILE,INPUTFILE,,1,,1024

0001+=            INPUTFILETYPE  EQU    INFILE
0103+C30F01        JMP      ??0009
0106+7E            MOV      A,M
0107+12            STAX     D
0108+23            INX      H
0109+13            INX      D
010A+0D            DCR      C
010B+C20601        JNZ      @DEF
010E+C9            RET
010F+215C00        LXI      H,@TCB+(1-1)*16
0112+111D01        LXI      D,FCBINPUTFILE
0115+0E0C            MVI      C,@C
0117+CD0601        CALL     @DEF
011A+C34401        JMP      ??000B
011D+              DS       @C
011D+=            FCBINPUTFILE EQU    $-12
0129+00            DB       0
.
.
.
.
.
NXTBLOCK:
01EF+CD4701        GET      INPUTFILE
                        CALL     GETINPUTFILE
01F2 CA0000        JZ       0000H
01F5 FE1A            CPI     1AH
01F7 CA0000        JZ       0000H

                        PUT      CON
01FA+F5            PUSH     PSW
01FB+0E02            MVI     C,@CON
01FD+5F            MOV      E,A
01FE+CD0500        CALL     @BDOS
0201+F1            POP      PSW

0202 C3EF01        JMP      NXTBLOCK

0205              DS       32
0225 =            STACK    EQU    $
                        BUFFERS:
0225              END

```

マクロ・アセンブラの実行により、マクロ・ライブラリ“SEQIO. LIB”から展開された部分。

A>

Figure-3.1.10 当プログラムの“PRN”ファイルのタイプアウト。マクロが展開されている様子がよく分かる。

Z80のアセンブル例

MACは、付属してくるマクロ・ライブラリの“Z80.LIB”を利用して、Z80用のソース・ファイルのアセンブルすることができます。使用されるニーモニックは、ザイログ表記ではなく、いわゆる“インテル拡張ニーモニック”と呼ばれるものです。

世の中にはいろいろな人がいて、“ザイログ表記を採用していないものは、Z80のアセンブラにあらず”と思っている人もいますが、アセンブラは、その総合的な機能と環境こそ第1に評価されるべきであり、その手段であるニーモニックの表現法のみを目を奪われるのはどうかと思います。筆者も、ザイログ表記の方がロジック的には分かり易いと思いますが、どちらにしても最重要な問題ではありません。

さて、MACによるZ80のアセンブル例を示します。

サンプル・プログラムとして、IF~ELSE~ENDIFの条件付きアセンブルの実習も兼ねて、1つのソース・ファイルで、8080、Z80の両方に対応できるプログラムを作ってみましょう。今回のプログラムは、メモリとCPUレジスタ間の操作を、8080とZ80の双方で、同じ内容のものを表記したもので、詳しくはリストをご覧ください。

そのソース・リスト（プログラム名は“8080Z80”）を次に示します。

```
A>TYPE 8080Z80.ASM! -----条件付きアセンブルによる、8080/Z80両用プログラム例のソース・ファイルの
                             タイプアウト。
*****
**          MAC SAMPLE PROGRAM 3          **
**          8080 & Z-80 USE              **
*****
TITLE      '--- 8080-Z80 SAMPLE PROGRAM ---'  -----TITLE宣言により、PRN!リストに
PAGE       50 -----PRN!リストを、1ページ50行に設定。      タイトル名が表示される。
ORG        100H
MACLIB     Z80 -----Z80マクロ・ライブラリの使用を宣言。このライブラリを使わなければ、Z80のアセン
                             ブルはできない。
TRUE       EQU      OFFFHH
FALSE      EQU      NOT TRUE
Z80        EQU      TRUE -----Z80用にアセンブルするため、こちらを真とする。
I8080      EQU      NOT Z80 -----反対に偽となる。

MVI        B,00

IF Z80
SETB       7,B -----Bレジスタのbit7 を1 にする。(Z80用ニモック)
ELSE
PUSH
MVI        A,1000$0000B
ORA        B
MOV        B,A
POP        PSW
ENDIF
Z80での“SETB7,B”と同じことを、8080で実現したもの。
```



```

LXI    H, 4004H  (レジスタに00をロードした後、bit7を1にして、アドレス4004Hにストア。)
MOV    M, B      (他のレジスタは変化させない。)

IF     Z80
  LHLD  0001H
  LDED  0006H  (アドレス0006H, 0007Hの内容をD, Eレジスタにロードする。(Z80用ニーモニック))
ENDIF

IF     I8080
  LHLD  0006H
  XCHG
  LHLD  0001H
ENDIF  (Z80での上記2ステップと同じことを、8080で実現したもの。)

IF     Z80
  SHLD  4001H
  SDED  4006H  (D, Eレジスタの内容を、アドレス4006H, 4007Hにストアする。(Z80用ニーモニック))
ELSE
  SHLD  4001H
  XCHG
  SHLD  4006H
ENDIF  (Z80での上記2ステップと同じことを、8080で実現したもの。)
      (アドレス0001H, 0002Hの内容を、4001H, 4002Hへ、アドレス0006H,
      0007Hの内容を4006H, 4007Hへコピーする。)

LXI    H, 3000H
LXI    D, 4080H
LXI    B, 128
CALL   BLKMOVE
RET

; <<<<< BLOCK MOVE SUB ROUTINE >>>>>
BLKMOVE:
IF     Z80
  LDIR  H, (レジスタでアドレスされるメモリ内容を、B, Cレジスタで示されるバイト数だけ、
  RET   D, (レジスタでアドレスされるメモリから順にブロック転送する(Z80用ニーモニック)).
ELSE
  MOV   A, B
  ORA   C
  RZ
  MOV   A, M
  STAX  D
  INX H! INX D
  DCX   B
  JMP   BLKMOVE
ENDIF  (Z80での上記2ステップと同じことを、8080で実現したもの。)
      (アドレス3000Hから128/バイトのデータを、アドレス4080Hからの)
      128/バイトにブロック転送する。
END

```

A>

Figure-3.1.11 Z80のアセンブル例のソース・リスト.

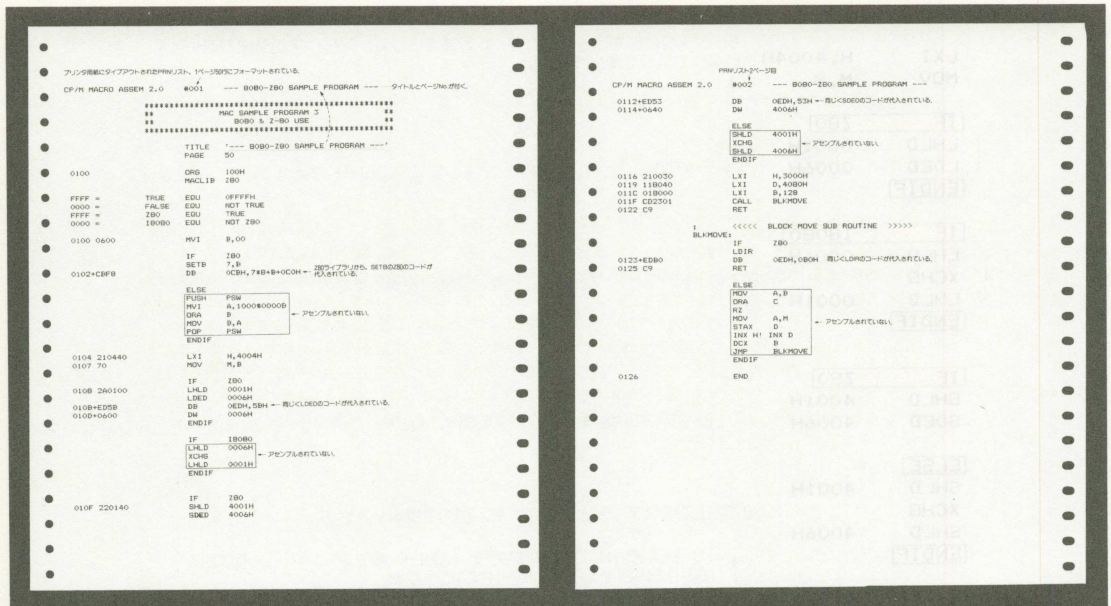


Figure-3.1.12 プリンタ用紙にタイプアウトされたサンプル・プログラムのPRNリスト。

上記のソース・ファイルをFigure-3.1.2に示したのと同様に、MACでアセンブルします。

次に、アセンブルにより生成されたPRNファイルをプリンタにタイプアウトして、そのリストを示します。“PAGE 50”の指定により、1ページ50行にリスト・フォーマットされています。

上記のPRNリストから、条件付きアセンブルが、どのように実行されているかが分かります。Z80が真、8080を偽としてアセンブルしたため、8080用の部分が無視されています。

アセンブルされている側の、8080のニーモニックにはないZ80ニーモニックの部分は、“Z80.LIB”マクロ・ライブラリからZ80のオブジェクト・コードが与えられています。

次に、当項の主旨ではありませんが、このサンプル・プログラムを実行してみましょう。先程のアセンブルによりHEXファイルも生成されていますので、LOADコマンドを実行して、“8080Z80.COM”ファイルを作ります。

では、実行前準備とプログラムの実行、それに結果の確認をした実行例を次に示します。

A>DDT JDDTを起動。

DDT VERS 2.2

-F3000,3100,55 Jアドレス3000H~3100Hに“55”をフィル。

-F4000,5000,00 Jアドレス4000H~5000Hを0クリア。あとで確認し易いように。

-^C DDTを終わる。

A>

[illegible]

Figure-3.1.13 サンプル・プログラムの実行前の準備と実行、それに結果の確認。

次に、参考までに、Figure-3.1.11の当プログラムのソース・リストの“Z80 TRUE”の部分を、“Z80 FALSE”と逆に書き直して、再アセンブルするとどうなるでしょう。

そのアセンブル後のPRNリストを次に示します。

```

CP/M MACRO ASSEMB 2.0      #001      --- BOB0-ZB0 SAMPLE PROGRAM ---
*****
**                               MAC SAMPLE PROGRAM 3                               **
**                               BOB0 B, 7-80 USE.                               **
*****
TITLE      "---- BOB0-ZB0 SAMPLE PROGRAM ----"
PAGE       30

0100      ORB      1004H
          MACLIB   ZB0

FFFF =     TRUE    EQU    OFFFHH
        =     FALSE EQU    NOT TRUE
        =     ZB0   EQU    FALSE
        =     FFFF  EQU    NOT ZB0

0100 0680      MVI      B,00

          IF      ZB0
          SETB    7,B

          ELSE
          PUSH    PSW
          MVI     A,1000H0000H
          ORA     B
          MOV     B,A
          POP     PSW
          ENDF

0100 210440     LXI     H,4004H
0100 70         MOV     H,B

          IF      ZB0
          LMD     0001H
          LDED    0006H
          ENDF

          IF      ZB0
          IF      ZB0
          010C 20A060     LMD     0006H
          010F EB        TCHS
          0110 20A010     LMD     0001H
          ENDF

          IF      ZB0
          SHLD   4001H
          SDED   4006H

          ELSE
          SHLD   4001H

0115 220140

```

Figure-3.1.14 先程のものと条件を逆にした場合のアセンブル結果.

このように今回は、8080用の部分がアセンブルされて、Z80用の部分が無視されています。

ここでも同じく、LOADコマンドで“8080Z80.COM”ファイルを作り実行すれば、今度は8080のオブジェクト・コードにより、Figure-3.1.13と同一の結果が得られます。

```
A>TYPE Z80.LIB/
;      @CHK MACRO USED FOR CHECKING 8 BIT DISPLACEMENTS
;
;@CHK   MACRO   ?DD      ;; USED FOR CHECKING RANGE OF 8-BIT DISP.S
;      IF (?DD GT 7FH) AND (?DD LT 0FFBH)
;      'DISPLACEMENT RANGE ERROR - Z80 LIB'
;      ENDIF
;      ENDM
LDX     MACRO   ?R,?D
;@CHK   ?D
;      DB      ODDH,?R*8+46H,?D
;      ENDM
LDY     MACRO   ?R,?D
;@CHK   ?D
;      DB      OFDH,?R*8+46H,?D
;      ENDM
;      }
LDI     MACRO
;      DB      OEDH,0A0H
;      ENDM
LDIR    MACRO
;      DB      OEDH,0B0H
;      ENDM
LDD     MACRO
;      DB      OEDH,0A8H
;      ENDM
;      }
```

Figure-3.1.14-a マクロ・ライブラリは、ユーザーが独自に変更したり、独自に開発したプログラムを新しいライブラリ・ファイルとしたり、自由に、使い易いようにすることができます。参考までに、“Z80.LIB”マクロ・ライブラリをタイプアウトして示しておきます。
“LDIR”は、Figure-3.1.11に使われていますので注目して下さい。

3.1.3 シンボル・テーブルについて

MACによるアセンブルの結果、シンボル・テーブルのファイルが生成されます。このファイルは、後述のSID（シンボリック・インストラクション・デバッグ）やZSIDのシンボル入力用としても使われます。

ここで、2.3章の“電子早見帳”のソース・ファイルをMACでアセンブルし、生成されたシンボル・テーブルを示します。

A>MAC RNDFILE /“電子早見帳”のソース・ファイルのアセンブル。

```
CP/M MACRO ASSEM 2.0
0434
004H USE FACTOR
END OF ASSEMBLY
```

A>DIR RNDFILE.* /アセンブルの終了後、生成された各ファイルの確認。

```
A: RNDFILE ASM : RNDFILE FRN : RNDFILE HEX : RNDFILE SYM
   ソース・ファイル
A>
```

Figure-3.1.15 2, 3章の“電子早見帳”のソース・ファイルをMACでアセンブルする。

上記のアセンブルで生成されたシンボル・テーブルをタイプアウトして次に示します。

各シンボルは、ABC順にソートされています。Figure-2.3.5のPRNリストの各アドレスと比較してみ下さい。

A>TYPE RNDFILE.SYM /生成されたシンボル・テーブルのタイプアウト。

Figure - 2.3.5のPRNリストの各アドレスを参照。

0005 BDOS	0279 CLOSE	010A CMDSEL	02C3 CONIN	02CB CONOUT
000D CR	0172 DATAIN	0181 DATAIN1	01EE DATAOUT	0080 DMA0
03B4 DMA1	01C5 DSEARCH	0270 ERROR	01FB EXIT	005C FCB
007D FCBRO	038D FLNEEDCLS	038C FLOPEN	0159 INSLMAP	0260 JUST
01A0 KEYIN	000A LF	0143 MAPWT	0387 MSGCRLF	0352 MSGDATIN
031E MSGERR	032A MSGIDXIN	02BD MSGOUT	02D4 MSGSEL	0366 MSGSRCH
012F NEWFILE	026D NOTFIND	0250 NXTIDX	021A OPNMPRD	0390 PMREC0
0392 PMRECC	038E PTIDAD	02A3 RNDREAD	02B0 RNDWRITE	01B9 RNDWT
024C SEARCH	0296 SEQWRITE	0287 SETRNDRC0	01CF SRCHIN	03B4 STACK

A>

Figure-3.1.16 生成されたシンボル・テーブルのタイプアウト。

3.1.4 SID, ZSIDの使用例

デジタルリサーチ社では、DDTよりさらに強力なシンボリック・インストラクション・デバッガの“SID”，および、Z80用のSIDである“ZSID”を用意しています。

ここで、ZSIDを起動して、先程のFigure-3.1.14～15のMACによるアセンブルで生成された電子早見帳の“RNDFILE”プログラムのHEXオブジェクト・ファイル“RNDFILE.HEX”をロードし、かつシンボル・テーブルも入力して、シンボリックにデバッグを行うことを想定して、2～3のコマンドを実行してみましよう。

SID, ZSIDは、DDTのコマンドとアップパーコンパチブルですので、もちろんDDTと同様なコマンドの使い方ができますが、ここでは、シンボルによるコマンド入力のみを2～3行ってみます。Figure-2.3.5の“RNDFILE”のリストと、Figure-3.1.16のシンボル・テーブルのリストとよく比較して下さい。

A>ZSID / ZSIDを起動. “#”がZSIDのプロンプトである.

ZSID VERS 1.4

#IRNDFILE.HEX RNDFILE.SYM / Iコマンドで、“RNDFILE”のHEXとSYMファイルをFCBに入力.

#R / Rコマンドでメモリに読み込む.

SYMBOLS シンボル・テーブルが読み込まれたことを示すメッセージ.

NEXT PC END

03BC 0100 8B6F

#L100 / Lコマンドで100Hから逆アセンブル.

0100 LD SP,03B4 .DMA1

0103 XOR A

0104 LD (03BC .FLOPEN),A

0107 LD (03BD .FLNEEDCLS),A

CMDSEL: ラベルが表示される.

010A LD DE,02D4 .MSGSEL

印に注目.

010D CALL 02BD .MSGOUT

表示されたアドレスが、シンボルのアドレスに当たっていれば、そのシンボル名を表示している.

0110 CALL 02C3 .CONIN

0113 CP 53

0115 JP Z,01C5 .DSEARCH

0118 CP 49

011A JP Z,0172 .DATAIN

#L.CONIN / Lコマンドでシンボル“CONIN”から逆アセンブル開始.

CONIN:

02C3 PUSH HL

02C4 LD C,01

02C6 CALL 0005 .BDOS

02C9 POP HL

02CA RET

CONOUT:

02CB PUSH HL

02CC LD C,02

02CE LD E,A

02CF CALL 0005 .BDOS

02D2 POP HL

02D3 RET

#D.DATAIN,.DSEARCH / Dコマンド、シンボル“DATAIN”から“DSEARCH”までをダンプ.

0172: 3A 8C 03 3C CA B1 01 CD 1A 02 3E FF 32 BD

. < > 2

0180: 03 3E FF CD 4C 02 3C CA 0A 01 11 2A 03 CD BD 02

. > L < *

0190: CD C3 02 2A 8E 03 77 11 52 03 CD BD 02 21 B4 03

. * w R !

01A0: CD C3 02 77 23 FE 1A CA B9 01 FE 0D C2 A0 01 3E

. w # >

01B0: 0A 77 CD CB 02 23 C3 A0 01 2A 92 03 22 7D 00 CD

. w # * " }

01C0: B0 02 C3 0A 01 3A

.


```

#XP /
P=0100 .DATAIN / プログラム・カウンタを、シンボル"DATAIN"にセット。
#X / 全レジスタ類の表示。 "DATAIN"のアドレスにセットされている。
----- A=00 B=0000 D=0000 H=0000 S=0100 P=0172
----- A=00 B=0000 D=0000 H=0000 X=0000 Y=0000 LD A, (038C .FLOPEN)
#^C ----- ZSIDを終りCP/Mへ。
A>

```

Figure-3.1.17 ZSIDの使用例。シンボルによるコマンド入力の場合。

このように、DDTでは絶対番地でアドレスの指定を行っていたものが、SID、ZSIDでは". symbol"と書くことにより、シンボル名によって、すべてのコマンドのアドレス指定ができます。もちろん絶対番地で行ってもかまいません。

3.2 RMACの使用例

3.2.1 RMACの機能

RMAC (Relocating Macro Assembler) は、前項で紹介したMACに、リロケートブルなオブジェクト・コードを発生する機能を加えたもので、次項で解説する"MACRO-80"とほとんど同じ機能を持っており、モジュール別の開発や各種コンパイラから、リロケートブル・オブジェクトとのリンク（後述）などが可能となるアセンブラです。リロケートブル・オブジェクトの形式は、次項で紹介する"MACRO-80"と同一であり、互いに互換性があります。

3.2.2 RMACの使い方実例

RMACによって可能となる、モジュール別マシン語開発法（1つのプログラムを、いくつかのモジュール〔部分〕に分け、それぞれのモジュール単位で責任を持って開発し、全部のモジュールの開発が終わった時点で、それぞれをリンク〔結合〕して1本のプログラムとする開発法）の紹介は、次項の"MACRO-80"の項で行いますので、その解説はここでは省略します（MACRO-80とほとんど同じ機能なので）。

RMACの実行例としては、次項の"モジュール別マシン語開発法"で実習するモジュールの1つを、このRMACを使ってアセンブルし、他のモジュールは、次項で行ったまま（MACRO-80でアセンブルしたもの）のものをを使い、全体をリンクしてプログラムを完成させてみましょう。

ただし、これらの解説は、次の「MACRO-80」の項を先に読まなければ理解しにくいので、まずは次項にジャンプして、それからここに帰って来て下さい。

アセンブラの実行とリンクの実行例

次項, 3.3章での実習プログラムで使した3つのモジュールの内,最後のモジュール"MODUL2"をRMACを使ってアセンブルし,次項と同様にリンクを行い,RMACとMACR-80により生成されるリロケートブル・オブジェクトが,同一形式であり,互換性があることを確認してみましょう。

RMACによってアセンブルされるソース・ファイルは,そのファイル名のエクステンションがMACと同じく"ASM"でなければなりません(MACRO-80では"MAC")。

よって,まず最後のモジュールのソース・ファイル名"MODUL2.MAC"を,"MODULX.ASM"とでもリネームしておいて下さい。他のモジュールはそのままの状態にしておきます。

アセンブルの実行では,"MODULX.ASM"を"RMACによりアセンブルします。このモジュールは,メッセージ用の文字列ファイルなので,8080,Z80には関係なく,Z80のライブラリを使用する必要はありません。

アセンブラの実行例を次に示します。

```
A>RMAC MODULX.I .....ソース・ファイル"MODULX.ASM"をアセンブル,
                        オプションのアセンブル・パラメータを付けない基本的なアセンブルを行う。
CP/M RMAC ASSEM 1.1
006B
000H USE FACTOR
END OF ASSEMBLY

A> アセンブルの終了。
```

Figure-3.2.1 RMACによるアセンブル実行例。

アセンブルが無事に終了しています。アセンブルにより,どのようなファイルが生成されたかをDIRにより次に示します。

```
A>DIR MODULX.*I ..... RMACによって生成された各種ファイルの確認。

A: MODULX PRN : MODULX ASM : MODULX REL : MODULX SYM
   リスト・ファイル   ソース・ファイル   リロケートブル・   シンボル・テーブル
A>                   オブジェクト・ファイル   ファイル
```

Figure-3.2.2 RMACにより生成された各種ファイルの確認。

生成されたファイルには、MACで生成される“HEX”ファイルはなく、代わりに“REL”と“SYM”の両ファイルが生成されています。“REL”はリロケートブル・オブジェクト、“SYM”はシンボル・テーブルの意味です。

次に、シンボル・テーブルの“MODULX.SYM”をタイプアウトして次に示します。

```
A>TYPE MODULX.SYM | -----シンボル・テーブルのリストアウト。
000D CR          000A LF          0037 MSGAFT          0000 MSGINP          001F MSGMNG
0050 MSGNIT
A>
```

Figure-3.2.3 RMACにより生成されたシンボル・テーブル。

それぞれのシンボルの相対アドレスは、次に示す“MODULX.PRN”リスト上のものと同じです。

```

PUBLIC MSGINP,MSGMNG,MSGAFT,MSGNIT

000D =      CR      EQU      0DH
000A =      LF      EQU      0AH

0000 0DOA4B4559 MSGINP: DB      CR,LF,'KEY INPUT ( M / A / N ) ? >#?'
001F 0DOA48454C MSGMNG: DB      CR,LF,'HELLO GOOD MORNING',CR,LF,'#'
0037 0DOA484920 MSGAFT: DB      CR,LF,'HI ! GOOD AFTERNOON',CR,LF,'#'
0050 0DOA474F4F MSGNIT: DB      CR,LF,'GOOD NIGHT BYE-BYE',CR,LF,'#'

006B                                END
```

Figure-3.2.4 RMACにより生成された“MODULX.PRN”のリストアウト。

PRNリストのページ割りは、MACと同様、“TITLE”や“PAGE”の指定により行われます。

アセンブルに際しての各種ファイルの入力先や、出力先、それに出力リストの形式や内容の指定などは、Figure-3.2.1のコマンド・ラインの後に、“アセンブル・パラメータ”を付けて実行することにより、MACと同様、いろいろなコントロールを行うことができますが、ここでは何も付けず、最も基本的なアセンブルを行いました。

リンク

RMACによって生成された、最後のモジュールのリロケートブル・オブジェクト・ファイルを、次項の実習ですでに生成されている他のモジュールとリンクし、1本のプログラムにしましょう。

その実行例を次に示します。

RMACで
アセンブル
したもの

① ② ③ ④

```
A>L80 MAIN,MODUL1,MODULX,MAINX/E/N/
```

-----L80の実行。①、②、③の各リロケートابل・オブジェクトのモジュールをリンクし、ファイル名④の“COM”ファイルを生成する。

```
Link-80 3.43 14-Apr-81 Copyright (c) 1981 Microsoft
```

```
Data 0100 01A7 < 167>
```

-----生成された純マシン・コードは、アドレス100H~1A7Hにロードされる167バイトの容量であることを示している。

```
32596 Bytes Free
```

-----残っている使用可能メモリ容量(この値はCP/Mのサイズによって異なる)。

```
[0000 01A7
```

↑

生成された純マシン・コードのページ数(1 ページは256バイト)。

```
A>DIR MAINX.*
```

-----生成されたCOMファイルの確認。

```
A: MAINX COM
```

-----ファイル名“MAINX”のCOMファイルが生成されている。

```
A>
```

Figure-3.2.5 RMACでアセンブルしたモジュールを、他のモジュールとリンクするリンカの実行。

これで最終的な実行可能のオブジェクト・ファイル、“MAINX.COM”ができ上がりました。さっそくこのプログラムを実行してみましょう。ただし、このプログラムは、Z80用に書かれていますので、8080や8085を使ったマシンの場合には、ソース・ファイルの変更が必要です。

でき上がったプログラムの実行例を次に示します。

```
A>MAINX
```

-----完成したプログラムの実行。

```
KEY INPUT ( M / A / N )? >M
HELLO GOOD MORNING
```

```
KEY INPUT ( M / A / N )? >A
HI ! GOOD AFTERNOON
```

```
KEY INPUT ( M / A / N )? >N
GOOD NIGHT BYE-BYE
```

```
KEY INPUT ( M / A / N )? >^Z
```

-----Ctrl-Zのキーインによりプログラムを終了してCP/Mに戻る。

```
A>
```

Figure-3.2.6 リンクにより完成したプログラムの実行。

3.3 MACRO-80によるモジュール別ソフト開発法とLINK-80

3.3.1 モジュール別ソフト開発法とは

ソフトウェアの開発において大きなシステムの開発になると、CP/Mに付属している“ASM”や、前述の“MAC”などのアブソリュートなアセンブラ（絶対番地のオブジェクト・コードを生成するアセンブラ）では、能率の良い開発を行うことが困難となり、今から解説を行う“MACRO-80”や前述の“RMAC”などのリロケートブルなアセンブラ（リロケートブルなオブジェクト・コードを生成するアセンブラ）が使われるようになります。つまり、大きなシステムを開発する場合、長大なプログラムを頭から順にコツコツと書いていったのでは、大変な時間がかかり、そのデバッグも容易なことではありません。

では、どうすればいいのでしょうか。そこで“モジュール別ソフト開発法”が登場してくる訳です。（この開発法は、マシン語に限らず高級言語についても同じです）。

大きなシステムを能率良く開発するには、全体のプログラムを、数個から数十個のモジュール（ブロック）に適切に分割します。そして、分割したそれぞれのモジュールに対し、入力条件、出力条件など、仕様を明確に定めた上でそれらのモジュールを複数の開発現場に分散し、それぞれを同時に平行して開発を進めるのです。

デバッグは、モジュール単位、あるいは関連あるいくつかのモジュールをリンクして行います。開発責任者は、すべてのモジュールの開発の進み具合を見ながら、すべてのモジュールが、ほぼ完成した適当な時期に、全体のモジュールをリンクして、総合的なデバッグに入ります。

このように、大きなプログラムをいくつかのモジュールに分割して、それぞれを独立に並列して開発を行い、最後に1つにまとめて完成させる方法が“モジュール別開発法”なのです。

次の図は、この“モジュール別開発法”の考え方のアウトラインを図示したものです。

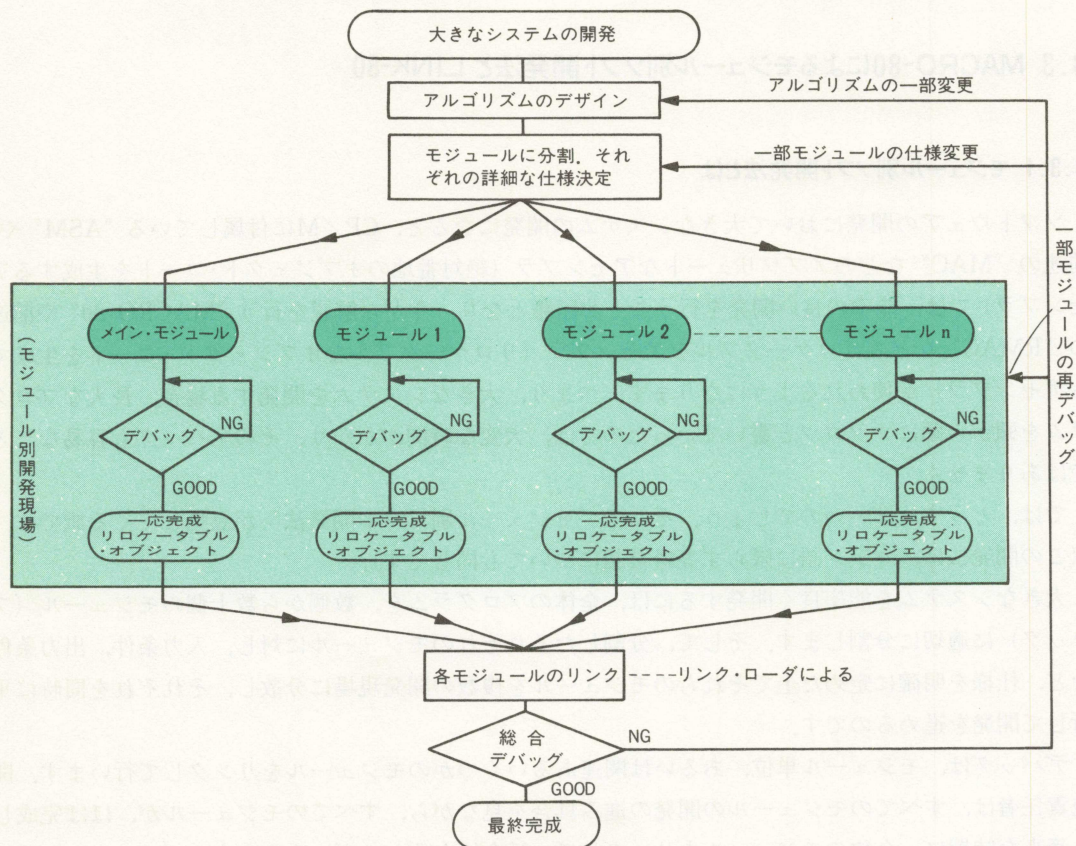


Figure-3.3.1 モジュール別ソフト開発の概念図

3.3.2 MACRO-80の機能

MACRO-80は、インテル形式の8080ニーモニック、およびザイログ形式のZ80ニーモニックをアセンブルし、リロケートブルなオブジェクト・ファイルと、PRNリスト・ファイル、それにシンボル・テーブルのリスト・ファイルを出力する8080/Z80両用のアセンブラで、マクロ機能を持っています。

MACRO-80により生成されるリロケートブル・オブジェクトの形式が、リロケートブル・オブジェクトの“標準フォーマット”となっており、3.2章のRMACなどの他社のアセンブラはもとより、5章で取り上げているような、各種高級言語のコンパイラ出力も、リロケートブルなものは、ほとんどがこの形式を採用しています(3.2章のRMACによるものとのリンク、3.5章のコンパイラとのリンクを参照)。

マクロ機能は、3.1章のMACとほぼ同じ機能を持っていますので、MACの項を参照して下さい(ただし、MACで可能なマクロ・ライブラリを取り込む機能は、MACRO-80にはありません)。

3.3.3 MACRO-80によるモジュール別ソフト開発の実例

では、モジュール別開発法により、1つの簡単なプログラムを作ってみましょう。

作成しようとするプログラム自身は、モジュール別開発を行うまでもない簡単なものですが、要は、その手法、開発過程に注目して下さい。

次のようなプログラムを作成します。

- プログラム名は "MAIN".
- プログラムを起動すると、「コマンドを入力せよ」というメッセージを表示する。
- そこで "M" をキーインすると「GOOD MORNING」
 "A" " 「GOOD AFTERNOON」
 "N" " 「GOOD NIGHT」
 と表示され、再びコマンド入力に帰る。
- プログラムの終了は、Ctrl-Zをキーインすることにより、CP/Mに戻る。

上記のプログラムを実現するために、プログラム全体を次に図示するように3つのモジュールに分け、それぞれについて別々に開発を行うことにしましょう。

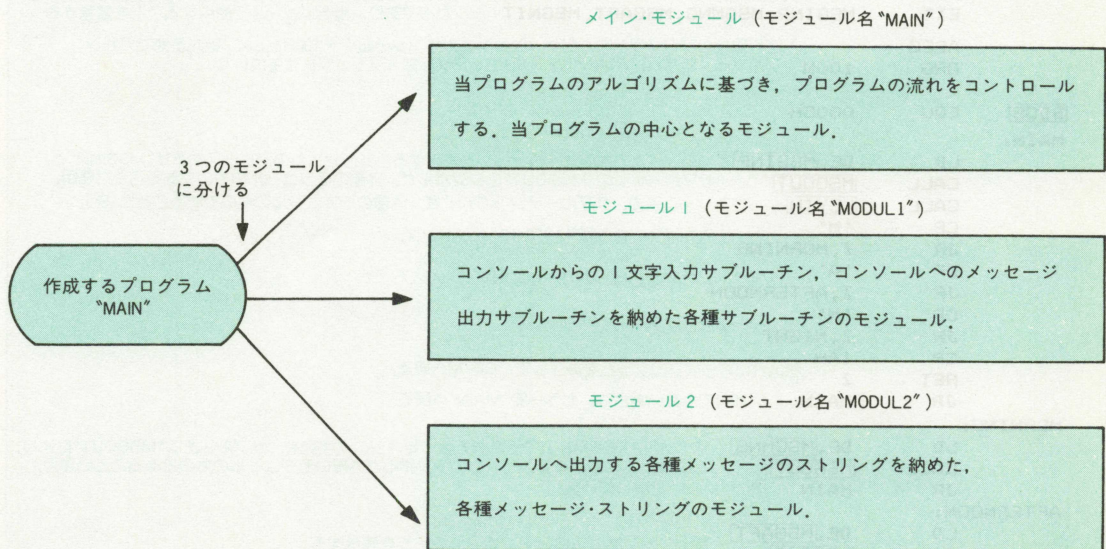


Figure-3.3.2 3つのモジュールに分けられた各モジュールの機能

ここでは、このように簡単にモジュールに分けることができましたが、実際の大きなシステムの開発となると、全体のアルゴリズムを考え、各モジュールに分け、それぞれの機能を特定する作業は、最も大変な作業となるでしょう。これができれば、もう開発は半ば完成したと言っても良いでしょう。後はただ、モジュール別の仕様に従って、それを満足するプログラムをひたすら組んで行けば良いのです。

各モジュール・レベルでの開発

さて、このプログラムのメイン・モジュールから順にプログラムを作って行きましょう。

メイン・モジュール

メイン・モジュールのソース・リストを次に示します。この場合、モジュール名を宣言する疑似命令の“NAME”を省略しましたので、モジュール名として、ファイル名が自動的に付けられます。

今回のプログラムは、Z80用にコーディングしてありますので、8080のマシン上でこの実習をする場合は、ソース・リストの一部を変更して下さい。

A>TYPE MAIN.MAC !-----メイン・モジュール、“MAIN”のソース・ファイルをタイプアウトして示す。

```

; *****
;      **      MACRO-80 SAMPLE PROGRAM      **
; *****
;
;      TITLE      '--- MACRO-80 SAMPLE PROGRAM ---'
;      PUBLIC BDOS ..... シンボル“BDOS”を、他のモジュールでも使用可能シンボルとして宣言。
;      EXT KEYIN,MSGOUT ..... これら5つのシンボルは、他のモジュールで宣言されている
;      EXT MSGINP,MSGMNG,MSGAFT,MSGNIT ..... シンボルであり、当モジュールで使用することを宣言する。
;
;      ASEG ..... これ以後のプログラムを、ロケーション・カウンタの起点を100Hとして、絶対番地でアセン
;      ORG 100H ..... プルする。つまり、“ASEG”は、1ロケータブルなオブジェクトは生成しない。
;
; [BDOS] EQU 0005H ..... システム・コールのエントリ・ポイント。
;
; MAIN:
;      LD DE,[MSGINP] ..... コマンド入力を促すメッセージを表示する。メッセージのアドレスを示す“MSGINP”と、
;      CALL MSGOUT ..... サブルーチンの“MSGOUT”のシンボルが、外部のモジュールのものであることに注目。
;      CALL KEYIN ..... コンソール入力。サブルーチン“KEYIN”が、外部のモジュールのものであることに注目。
;      CP 'M' ..... 入力“M”ならば“MORNIG”へジャンプ。
;      JR Z,MORNING
;      CP 'A' ..... “A”ならば“AFTERNOON”へジャンプ。
;      JR Z,AFTERNOON
;      CP 'N' ..... “N”ならば“NIGHT”へジャンプ。
;      JR Z,NIGHT
;      CP 1AH ..... Ctrl-Zならばプログラムを終了して、CP/Mへ戻る。
;      RET Z
;      JR MAIN ..... いずれでもない場合は、もう一度“MAIN”へ戻る。
;
; MORNING:
;      LD DE,[MSGMNG] ..... シンボル“MSGMNG”で示されるメッセージの文字列を、サブルーチン“MSGOUT”により、
;      CALL MSGOUT ..... コンソールに表示する。これらのシンボルが外部のモジュールのものであることに注目。
;      JR MAIN ..... 再び“MAIN”に戻り繰り返し。
;
; AFTERNOON:
;      LD DE,[MSGAFT] ..... 同上。シンボル“MSGAFT”のメッセージを表示する。
;      CALL MSGOUT
;      JR MAIN

```



```

NIGHT:
      LD      DE,MSGNIT
      CALL   MSGOUT
      JR      MAIN
      END
A>

```

同上。シンボル"MSGNIT"のメッセージを表示する。

Figure-3.3.3 メイン・モジュール "MAIN" のソース・リスト。

上記ソース・リストを基に、エディタを使ってソース・ファイルを作ります。ファイル名は"MAIN. MAC" としました。

MACRO-80でアセンブルするソース・ファイルのエクステンションは、必ず"MAC" でなければなりません。

ソース・ファイルが作成できたら、次はアセンブルを実行します。M80 (MACRO-80のコマンド・ファイル名) を実行する際のコマンド・ラインにより、前述のMACやRMACと同じように、入力ファイルや出力ファイルの指定や、アセンブル時の各種のコントロールを行うことができますが、ここでは最も基本的な形式で実行します。

M80によるソース・ファイル "MAIN. MAC" のアセンブル例を次に示します。

```

      "REL" "PRN" ソース
      ファイル名 ファイル名 ファイル名
A>M80 MAIN,MAIN=MAIN/Z  Z80用のソース・ファイルをアセンブルするためのスイッチ。
      .....ソース・ファイル"MAIN. MAC"のアセンブル。

No Fatal error(s) .....アセンブルの終了。

A>DIR MAIN.*! .....生成された各ファイルの確認。
A: MAIN      MAC : MAIN      PRN : MAIN      REL
      ↑           ↑           ↑
A>   ソース・ファイル   PRNリスト・ファイル   リロケートابل・
                                   オブジェクト・ファイル

```

Figure-3.3.4 M80によるソース・ファイル "MAIN. MAC" のアセンブルと、その結果の確認。

このように、ソース・ファイル "MAIN. MAC" から、リスト・ファイル "MAIN. PRN" と、リロケートابلのオブジェクト・ファイル "MAIN. REL" が生成されています。

次に、このPRNリスト・ファイルをプリンタにタイプアウトしてみましょう。プリント用紙の折り目を適当な位置にセットして、

```
A>TYPE MAIN. PRN ^P ]
```

を実行することにより、プリンタにPRNリストを出力できます。

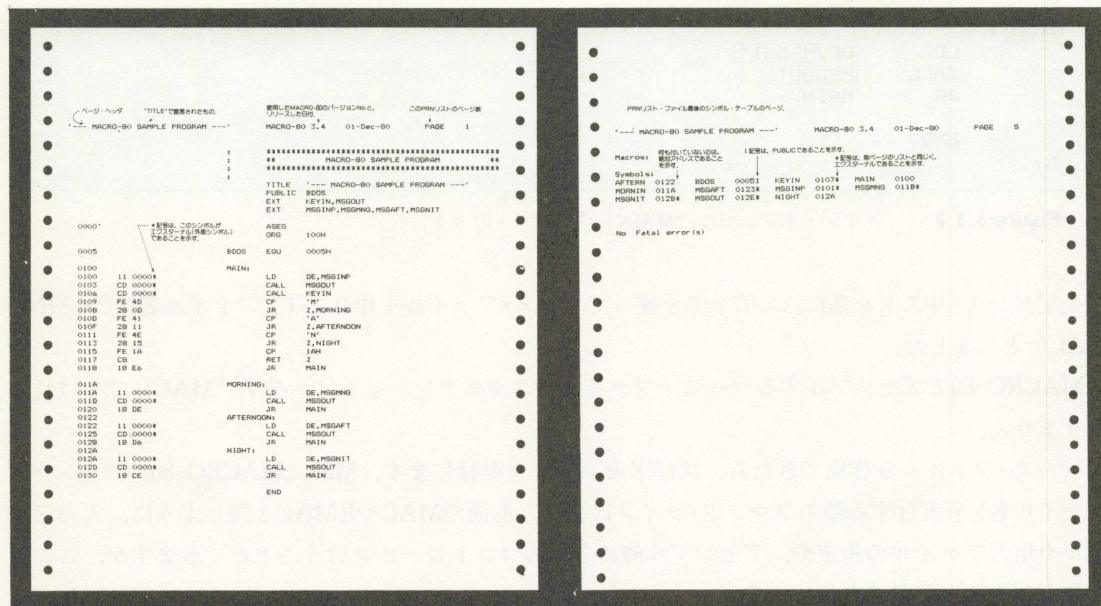


Figure-3.3.5 アセンブルにより生成されたPRNリスト・ファイルのプリンタへのタイプアウト。

このように、PRNリスト・ファイルは、1 ページ50行のページ割り付けが行われ(疑似命令“PAGE”の指定により、任意の行数に設定することが可能)、ソース・ファイルで宣言した“TITLE”により、ページ・ヘッダ部にタイトル名とページNoが表示されます。PRNリスト最後の“PAGE S”のページは、シンボル・テーブルがリストアウトされるページです。

モジュール 1

メイン・モジュールの場合と同様に、ソース・リスト、アセンブラの実行、生成されたPRNファイルのタイプアウトを示します。実行方法などは、メイン・モジュールの場合と同様ですので省略します。

モジュール1のソース・ファイル (MODUL1.MAC) のリスト.

```
A>TYPE  MODUL1.MAC/
```

PUBLIC	KEYIN, MSGOUT	-----この2つのシンボルを、他のモジュールでも使用可能シンボルとして宣言。
EXT	BDOS	-----他のモジュールのシンボル"BDOS"を使用することを宣言。

KEY IN:

LD C, 1
CALL BDOS
RET

コンソールからの1文字入力のサブルーチン。


```
MSGOUT:
  LD      C,9
  CALL    BDOS
  RET
END
```

メッセージ出力のサブルーチン。

A>

Figure-3.3.6 モジュール1 (MODUL1.MAC) のソース・リスト。

ソース・ファイル "MODUL1.MAC" のアセンブル実行。

```
A>M80 MODUL1,MODUL1=MODUL1/Z/ .....MODUL1.MACをアセンブル。Z80用なので"/Z"スイッチを付ける。
No Fatal error(s)

A>DIR MODUL1.* / .....生成された各ファイルの確認。
A: MODUL1 PRN : MODUL1 MAC : MODUL1 REL
      ソース・ファイル
A>
```

Figure-3.3.7 ソース・ファイル "MODUL1.MAC" のアセンブルと、生成された各ファイルの確認。

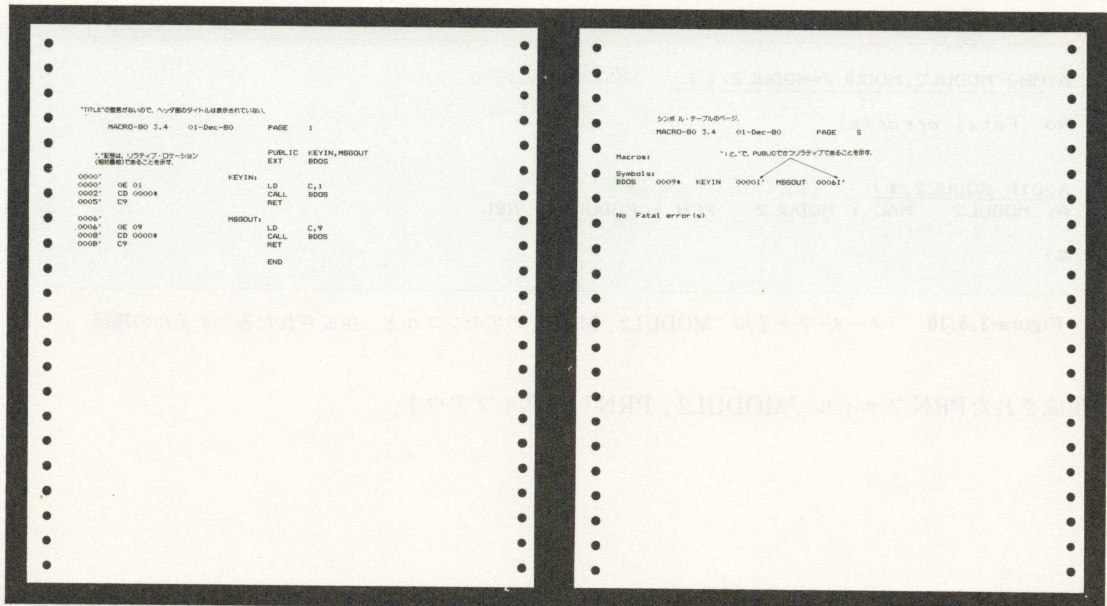


Figure-3.3.8 生成されたPRNファイル "MODUL1.PRN" のタイプアウト。

モジュール 2

モジュール 2 の場合も同様に、ソース・リスト、アセンブラの実行、生成された PRN ファイルのタイプアウトを示します。

モジュールのソース・ファイル (MODUL2. MAC) のリスト。

A>TYPE MODUL2.MAC /

PUBLIC MSGINF,MSGMNG,MSGAFT,MSGNIT4つのシンボルにPUBLICを宣言。

CR EQU 0DH
LF EQU 0AH

MSGINF: DB CR,LF,'KEY INPUT (M / A / N)? >\$'
MSGMNG: DB CR,LF,'HELLO GOOD MORNING',CR,LF,'\$'
MSGAFT: DB CR,LF,'HI ! GOOD AFTERNOON',CR,LF,'\$'
MSGNIT: DB CR,LF,'GOOD NIGHT BYE-BYE',CR,LF,'\$'

各メッセージのストリング。

END

A>

Figure-3.3.9 モジュール 2 (MODUL2. MAC) のソース・リスト。

ソース・ファイル "MODUL2. MAC" のアセンブル実行。

A>MB0 MODUL2,MODUL2=MODUL2/Z /MODUL2.MACをアセンブル。

No Fatal error(s)

A>DIR MODUL2.* /生成された各ファイルの確認。

A: MODUL2 MAC : MODUL2 PRN : MODUL2 REL
ソース・ファイル

A>

Figure-3.3.10 ソース・ファイル "MODUL2. MAC" のアセンブルと、生成された各ファイルの確認。

生成された PRN ファイル "MODUL2. PRN" のタイプアウト。

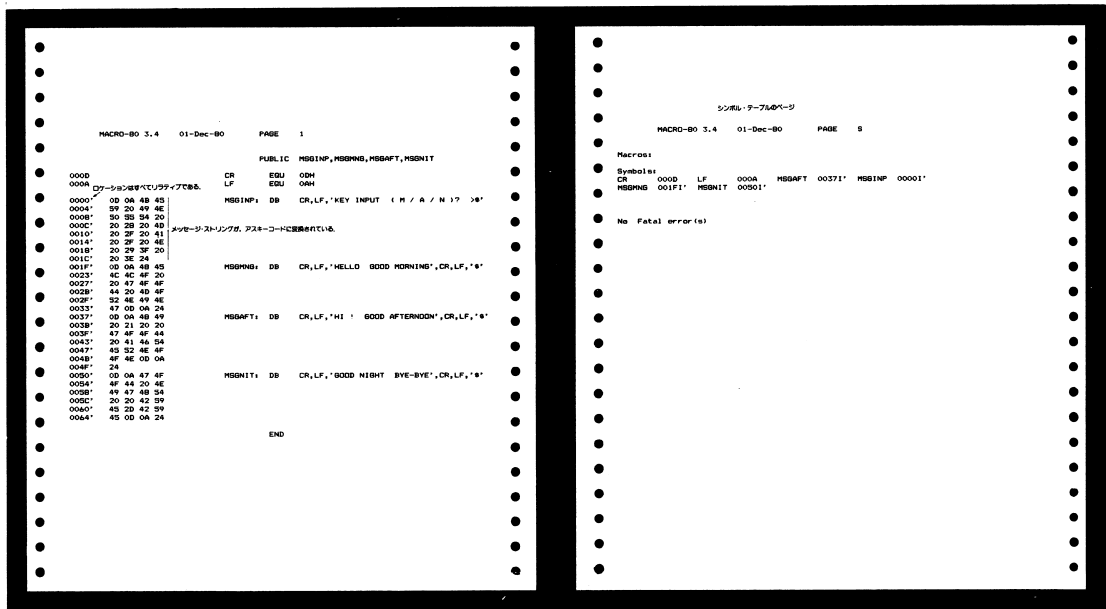


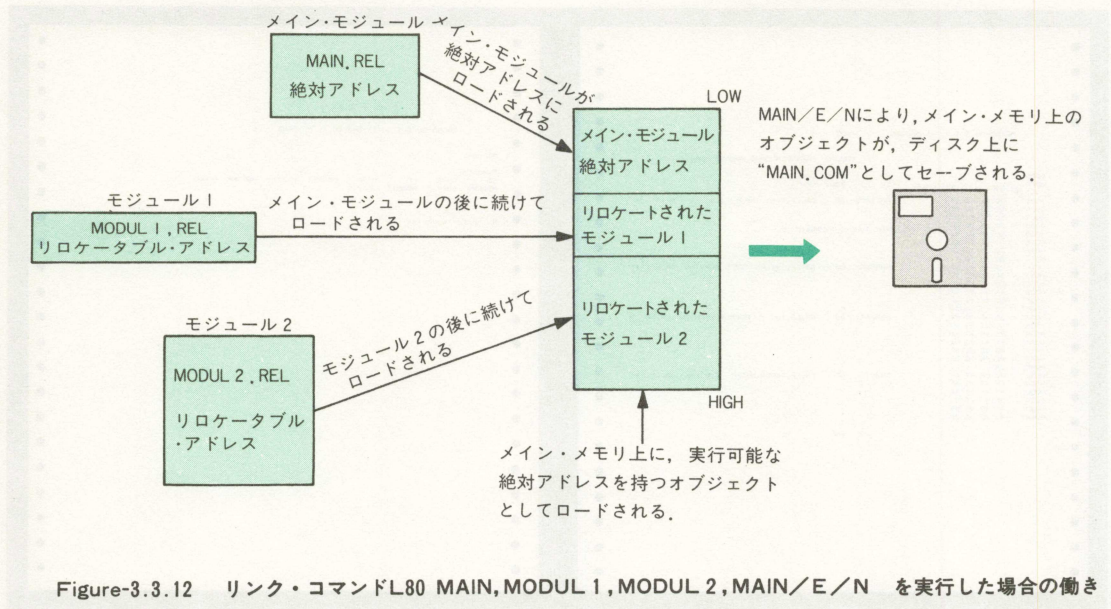
Figure-3.3.11 生成されたPRNファイルのタイプアウト。

さて、以上で「メイン・モジュール」、「モジュール1」、「モジュール2」のすべてのモジュールの開発が一応終わりました。本来ならば、モジュール単独でのデバッグを行う訳ですが、ここでは省略しています。次のステップは、いよいよ「リンク」です。

LINK-80によるオブジェクト・モジュールのリンク

リンク・ローダは、今までバラバラに開発を行ってきて、でき上った各モジュールのリロケータブル・オブジェクト・ファイル（例えば、MAIN.RELや、MODUL1.RELなど）を、任意の順序でつなぎ合わせ、実行可能な絶対アドレスを持った1本のオブジェクト・ファイル（COMファイル）に変換するものです。

今回の例では、次の図に示すようなことを行います。



では、LINK-80 (コマンド・ファイル名 "L80.COM") を実行してみましょう。

L80も実行時のコマンド・ラインにオプションの "スイッチ" を付けることにより、ローディングに際し、各種の処理を行わせることができます。

今回のリンク実行時のコマンド・ラインの意味は、

1. L80を起動し、
2. ディスク上の "MAIN.REL" を実行可能なオブジェクト・コードとしてロードし、
3. それに続けて、"MODUL1.REL" と "MODUL2.REL" を同様にロードし、
4. メモリ上にロードし終わった、実行可能なオブジェクトを "MAIN.COM" というファイル名でディスク上にセーブし、
5. セーブが終了した後、CP/Mに戻る。

という内容を、L80に指示しています。

その実行例を次に示します。

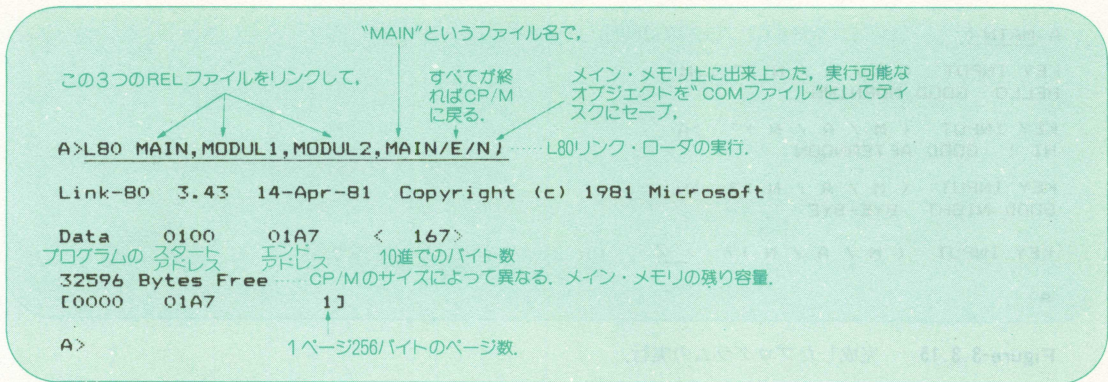


Figure-3.3.13 L80によるリンク・ローダの実行例。

この場合、メイン・モジュールは、"ASEG"と"ORG 100H"により、先頭のロケーションは100Hに指定されており、L80のコマンドで特に指定しなくても、"MAIN.REL"は自動的に100Hからロードされます。

他のモジュールは、"CSEG"指定(ソース・ファイルに、ASEG, CSEG, DSEGなどの指定を省略した場合は、自動的にCSEG指定となる)なので、リロケートブルであり、それぞれ前モジュールの最後のアドレスに続いてロードされます。

次に、生成されたCOMファイルをDIRで確認してみましょう。

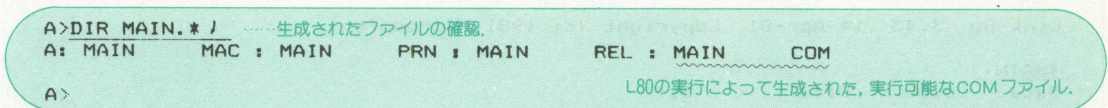


Figure-3.3.14 L80により生成されたCOMファイルの確認。

このように、3つのモジュールの"REL"オブジェクト・ファイルから、実行可能な1本の"COM"ファイルが生成されていることが確認されます。

リンクされたプログラムの実行

リンクが終わると、次はプログラム全体をテストランさせて、総合的なデバッグに入ります。バグがあれば、該当モジュールの修正→再リンクを繰り返し行います。

当プログラムは単純なので、デバッグの必要もない程ですが、一応、実動テストをしてみましょう。

当プログラム"MAIN.COM"の実行例を次に示します。


```
A>MAIN / ..... リンクして完成されたプログラムの実行.
KEY INPUT ( M / A / N )? >M
HELLO GOOD MORNING
KEY INPUT ( M / A / N )? >A
HI ! GOOD AFTERNOON
KEY INPUT ( M / A / N )? >N
GOOD NIGHT BYE-BYE
KEY INPUT ( M / A / N )? >^Z ..... Ctrl-Zのキーインにより、プログラムを
                                     終了してCP/Mに戻る.
A>
```

Figure-3.3.15 完成したプログラムの実行.

このように、プログラムは完動し、今回の開発はめでたく終了ということになりました。

リンク・ローダのその他の解説

L80の実行は、Figure-3.3.13に示されている例のほか、次のように、モジュールを1つ1つ入力していってもリンクすることができます。モジュールがリンクされるたびに、最終アドレスやオブジェクト・サイズを表す値が増加していくので、リンクの動作を実感することができますので、試みて下さい。その実行例を次に示します。

```
A>LB0 / ..... L80を起動.
Link-80 3.43 14-Apr-81 Copyright (c) 1981 Microsoft
*MAIN / ..... メイン・モジュールを入力.
Data 0100 0133 < 51>
-KEYIN 0107 -MSGAFT 0123 -MSGINP 0101
-MSGMNG 011B -MSGNIT 012B -MSGOUT 012E
6 Undefined Global(s)
32712 Bytes Free
*MODUL1 / ..... モジュール1を入力.
Data 0100 013F < 63>
-MSGAFT 0123 -MSGINP 0101 -MSGMNG 011B
-MSGNIT 012B
4 Undefined Global(s)
32700 Bytes Free
*MODUL2 / ..... モジュール2を入力.
Data 0100 01A7 < 167>
32596 Bytes Free
```



```

*MAIN/E/N / ..... メモリ上にリンクされた、今までのモジュールのオブジェクトを"MAIN.COM"というファイル名でディ
                     スクにセーブし、CP/Mに戻る。

Data      0100      01A7      < 167>

32596 Bytes Free
[0000      01A7      11

A>

```

Figure-3.3.16 L80による別の方法でのリンク。

この方法でリンク・ローダを実行しても、前回と全く同様な“COM”ファイルが生成され、プログラムは実行します。

Figure-3.3.13や、上記のL80の実行により生成された実行可能なオブジェクト・ファイルを、DDTを使ってダンプして次に示します。

```

A>DDT MAIN.COM / ..... DDTを起動して、“MAIN.COM”をロードする。

DDT VERS 2.2
NEXT PC
0200 0100

L80が挿入したスペース・コード。
メイン・モジュール。
モジュール1。

-D100
0100 11 3F 01 CD 39 01 CD 33 01 FE 4D 2B 0D FE 41 2B .?.?.3..M(..A(
0110 11 FE 4E 2B 15 FE 1A CB 18 E6 11 5E 01 CD 39 01 ..N(.....^..9.
0120 18 DE 11 76 01 CD 39 01 18 D6 11 8F 01 CD 39 01 ...v..9.....9.
0130 18 CE 20 0E 01 CD 05 00 C9 0E 09 CD 05 00 C9 0D .. .....
0140 0A 4B 45 59 20 49 4E 50 55 54 20 20 2B 20 4D 20 .KEY INPUT ( M
0150 2F 20 41 20 2F 20 4E 20 29 3F 20 20 3E 24 0D 0A / A / N ) ? > $ ..
0160 4B 45 4C 4C 4F 20 20 47 4F 4F 44 20 4D 4F 52 4E HELLO GOOD MORN
0170 49 4E 47 0D 0A 24 0D 0A 4B 49 20 21 20 20 47 4F ING..$..HI ! GO
0180 4F 44 20 41 46 54 45 52 4E 4F 4F 4E 0D 0A 24 0D 0D AFTERNOON..$.
0190 0A 47 4F 4F 44 20 4E 49 47 4B 54 20 20 42 59 45 .GOOD NIGHT BYE
01A0 2D 42 59 45 0D 0A 24 01 18 D6 11 8F 01 CD 39 01 -BYE..$.....9.
01B0 18 CE 20 0E 01 CD 05 00 C9 0E 09 CD 05 00 C9 0D .. .....
-^C
A>
モジュール2。

```

Figure-3.3.17 完成した実行可能オブジェクト・ファイルのダンプ。

3つのモジュールが、上記のように配置されていることが分かります。

次に、今までは、アドレス100Hをスタート・アドレスとしてリンクを行ってきましたが、任意のアドレスをスタート・アドレスとしてロードするにはどうすれば良いでしょう。その実例を示します。

まず、メイン・モジュールのソース・ファイルに書かれている、“ASEG”と“ORG 100H”のスタート・アドレスを絶対番地の100Hに固定する2つの疑似命令を、Figure-3.3.3のソース・リスト“MAIN.MAC”から削除し、ファイル名を変えて“MAINXADR.MAC”としておきます。その後、Figure-3.3.4と同様にアセンブルして、“MAINXADR.REL”を生成しておきます。その時に生成されたPRN

ファイルをタイプアウトして、その1ページ目を次に示します。Figure-3.3.5のPRNリストと比較して下さい。

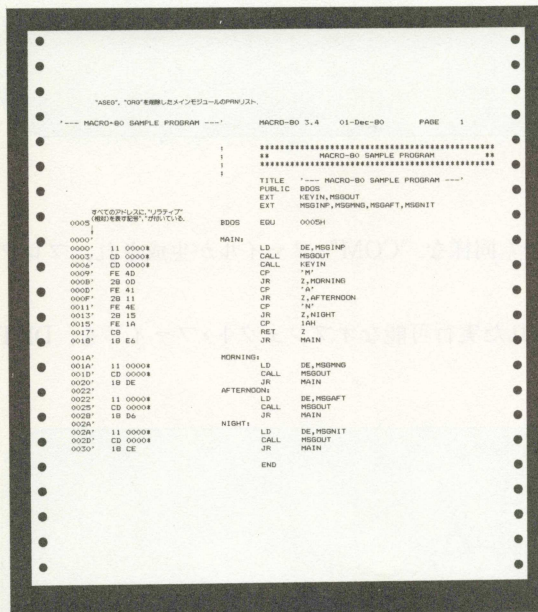


Figure-3.3.18 "ASEG", "ORG" を削除したメイン・モジュールのPRNリスト。

さて、任意のアドレスへリロケートする準備ができたので、プログラムのスタート・アドレス（メイン・モジュールの先頭アドレス）を4000Hとして、L80を実行してみましょう。その実行例を次に示します。

A>L80 / L80を起動。

Link-80 3.43 14-Apr-81 Copyright (c) 1981 Microsoft

* /P:4000,MAINXADR,MODUL1,MODUL2/E / 最初のモジュールの先頭アドレスを4000Hにセットして、続くモジュールを順にメモリ上にロードして行く。最後は/EでCP/Mに戻る。

Data 4000 40A6 < 166>

32597 Bytes Free

[0000 40A6 64]

A>

Figure-3.3.19 プログラムのスタート・アドレスを4000HとしてL80を実行。

上記コマンドのスイッチ "/P:4000" により、最初のモジュールの先頭は4000Hにロードされたはずです。

DDTで全体のロード状態を確認してみましょう。その実行例を次に示します。

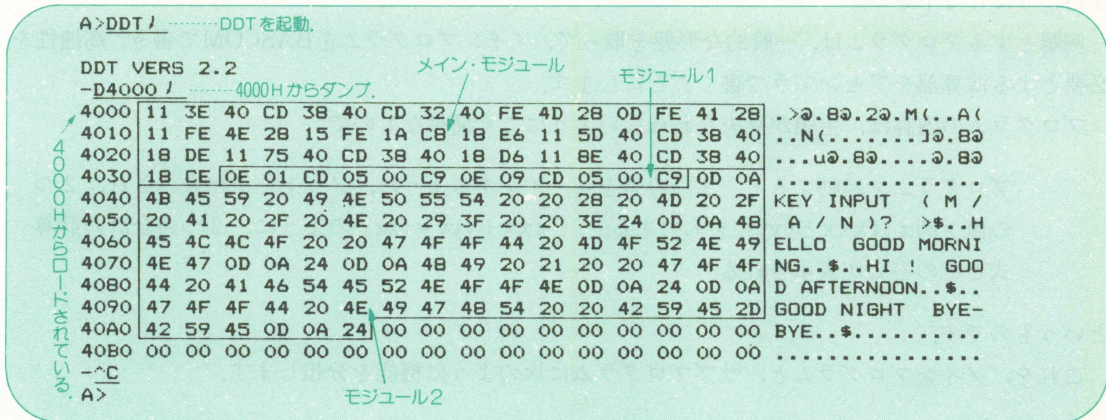


Figure-3.3.20 DDTでアドレス4000Hからのロード状態を確認する。

このように、リンクは先頭を4000Hとして行われていることが確認できます。もちろん、このプログラムは4000Hスタートで実行することができます。上記DDTを終わる前にコマンド「G4000!」により、Figure-3.3.15と同様に、当「MAIN」プログラムを実行することができます。試みて下さい。

3.4 高級言語コンパイラとマシン語とのリンク

本書の5章でも、その何種類かを取り上げていますが、コンパイラ型の高級言語の多くは、マイクロソフト社フォーマットのリロケータブル・オブジェクト・ファイルを出力します。

よって、メインプログラムを記述性・保守性のよいコンパイラで作成し、高速を要求されるものや、アセンブラでなければ記述が困難な部分をアセンブラで作成して、両者をリンク・ローダでリンクして、1本のプログラムとすることができます（コンパイラ側がメインプログラムでなければならない訳ではありません）。

また、コンパイラとアセンブラのリンクに限らず、例えば、COBOLからのリロケータブル・オブジェクトと、PL/Iからのリロケータブル・オブジェクトなど、コンパイラとコンパイラとをリンクすることも可能です。

実例として、BASICコンパイラ（マイクロソフト社のBASIC COMPILER "BASCOM"）とアセンブラとのリンクを示します。高級言語とマシン語とのリンクと言っても、一般のパーソナル・コンピュータで使われているBASICインタプリタからCALLやUSERなどでマシン語を呼ぶことは、基本的に異なり、ここで解説しているのは高級言語で作られたマシン・コードと、アセンブラで作られた

マシン・コードを、マシン・コード同志リンクして、1本のプログラムにすることですので、誤解しないようにして下さい。

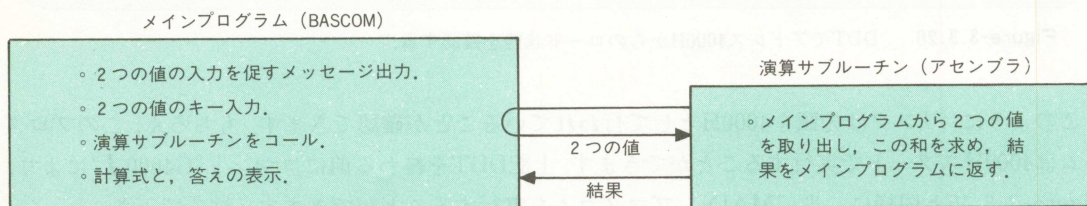
例題とするプログラムは、一般的な形態を取って、メインプログラムをBASCOMで書き、高速性を必要とする演算部をアセンブラで書くことにします。

プログラムの内容は、主題が見失なわれないようにごく簡単なもので、

プログラムを起動すると、2つの値をキー入力するように、メッセージが表示され、2つの値（例えばxxxとyyy）を入力すると、「xxx+yyy=zzz」のように、2つの合計の計算式とその答えが表示される。

というものです。

これを、メインプログラムと、サブプログラムに次のように機能を分担します。



3.4.1 BASCOMによるメインプログラムの作成

メインプログラムのBASCOMのソース・リストを次に示します。このソース・ファイル“HIGHLANG.BAS”を、CP/Mのエディタ、あるいはMBASICを持っている方はMBASIC上で作成します。

内容は単純なので特に解説の必要はないでしょう。プログラムを簡単にするため、数値を2バイトの整数として取り扱うので、“DEFINT”宣言をしています。

BASICインタプリタでは、“CALL”ステートメントのアドレス値が宣言されていなければなりませんが、BASCOMでは、宣言されていない場合は自動的に“EXT”シンボルとして処理されます（3.2および3.3章参照）。

A>TYPE HIGHLANG.BAS) -----BASCOMによるメインプログラムのソース・ファイルのタイプアウト。

```
1000 *****
1010 ***   MAIN ROUTINE BY HIGH LEVEL LANGUAGE   ***
1020 *****
1030 '
1040 DEFINT A-Z -----数値が2 バイトで表される。プログラムを簡単にするため。
1050 '
1060 RESULT=0 -----演算結果を格納するアドレスを知るためのダミー。
```



```

1070      PRINT: PRINT "INPUT PARAMETER 1 AND 2, ( parm1,parm2 ) >";
1080      INPUT PARAMETER1,PARAMETER2
1090      '
1100      CALL MCNCAL (PARAMETER1,PARAMETER2,RESULT)
1110      '
1120      PRINT: PRINT PARAMETER1;" + ";PARAMETER2;" = ";RESULT
1130      GOTO 1060
1140 END

```

HLLレジスタに、DELレジスタに、BCLレジスタに。
 自動的にシンボル"MCNCAL"として"EXT"処理される。

A>

Figure-3.4.1 BASCOMによるメインプログラムのソース・リスト、ファイル名のエクステンションは“BAS”であること。

次にこれをコンパイルします。

BASCOMは、バージョン5.30からランタイム・ライブラリの使い方が変更され、通常のコンパイルでは、ランタイム・ライブラリの一部は、生成される“REL”オブジェクトに組み込まれず、外部ライブラリとして、プログラムの実行時に使われることになりました。ただし、プログラムをROM化する場合や、完全に1本のプログラムとする場合は、ランタイム・ライブラリを全部オブジェクトに組み込む必要があります、そのスイッチが実行例に示す“/O”なのです（このバージョンから、カナも使えるようになりました）。

BASCOMの実行例を次に示します。

A>BASCOM HIGHLANG,HIGHLANG=HIGHLANG/O !コンパイルの実行("/O"スイッチ本文参照)。

```

00000 Fatal Error(s)
10586 Bytes Free
コンパイル終了。

```

A>DIR HIGHLANG.* !コンパイルにより生成された各ファイルの確認。

A: HIGHLANG BAS : HIGHLANG PRN : HIGHLANG REL

A> ソース・ファイル PRNファイル リロケートابل・オブジェクト・ファイル

Figure-3.4.2 BASCOMによるコンパイル実行例と生成された各ファイルの確認。

以上でメインプログラムのリロケートابل・オブジェクトファイルができました。参考までに、生成されたPRNファイルをタイプアウトして示します。BASIC言語のソースが、アセンブラのソース・レベルに展開されている（落ちている）ことが分かります。

A>TYPE HIGHLANG.PRN)PRNファイルのタイプアウト.

BASCOM 5.30 - Copyright 1979,80,81 (C) by MICROSOFT - 10962 Bytes Free

```
0014 0007      1000 '*****  
0014 0007      1010 '***   MAIN ROUTINE BY HIGH LEVEL LANGUAGE   ***  
0014 0007      1020 '*****  
0014 0007      1030 '  
0014 0007      1040 DEFINT A-Z  
0014 0007      1050 '  
0014 0007      1060 RESULT=0  
        ** 0014'I00000: CALL    $530  
        ** 0017'L01000: L01010: L01020: L01030: L01040: L01050: L01060:  
        ** 0017'       LXI      H,0000  
        ** 001A'       SHLD     RESULT%  
001D 0009          1070             PRINT: PRINT "INPUT PARAMETER 1 AND 2, ( parm1,param2 ) >";  
        ** 001D'L01070: CALL    $PROA  
        ** 0020'       LXI      H,<const>  
        ** 0023'       CALL    $PV2D  
.  
.  このようにアセンブラのソース・レベルに変換されている。  
.
```

Figure-3.4.3 コンパイルにより生成されたPRNファイルのタイプアウト.

3.4.2 アセンブラによる演算ルーチンの作成

BASCOMの“CALL”ステートメントにより、それぞれの変数のアドレスが、PARAMETER1は(HL)、PARAMETER2は(DE)、RESULTは(BC)の各レジスタにより与えられます。よって当演算ルーチンは、それらの値を取り出してその和を求め、RESULTのアドレスへ答えを格納すれば良いのです。極めて簡単な“DAD”を使った16ビットの和を求める演算ルーチンのソース・リスト(アセンブルにより生成されたPRNファイル)を次に示します。これを基に、ソース・ファイル“MACHINE.MAC”をエディタで作成して下さい。

アセンブラによる演算サブルーチンのアセンブルにより生成された、PRNソース・リスト。
MACRO-80 3.4 01-Dec-80 PAGE 1

MACRO-80 3.4 01-Dec-80 PAGE 5

```

;*****
;**      MACHINE SUB ROUTINE      **
;*****

アセンブル後のオブジェクトは、すべてリロケートラブルであることを示す。

0000'
0000' C5
0001' 4E
0002' 23
0003' 46
0004' EB
0005' 5E
0006' 23
0007' 56
0008' EB
0009' 09
000A' C1
000B' 7D
000C' 02
000D' 03
000E' 7C
000F' 02
0010' C9

```

```

PUBLIC MCNCAL .....シンボル"MCNCAL"を
PUBLIC宣言。

MCNCAL:
PUSH B
MOV C,M
INX H
MOV B,M
XCHG
MOV E,M
INX H
MOV D,M
XCHG
DAD B
POP B
MOV A,L
STAX B
INX B
MOV A,H
STAX B
RET

```

PARAMETER1の値をB、
Cレジスタにロード。

PARAMETER2の値をD、
Eレジスタにロード。

PARAMETER1+PARAMETER2
の値をH、Lレジスタにロード。

与えるH、Lレジスタの値を、
"RESULT"のアドレスへストア。

END演算サブルーチン終わり、リターン。

Macros:

Symbols:

MCNCAL 0000I'

No Fatal error(s)

Figure-3.4.4 アセンブラによる演算ルーチンのソース・リスト (アセンブルにより生成されたPRNファイル).

ソース・ファイル "MACHINE.MAC" を、M80あるいはRMACでアセンブルします（実行例は省略、3.2、3.3章参照）。

アセンブルにより、演算ルーチンのオブジェクト・ファイル "MACHINE.REL" が生成されたら、いよいよメインプログラムとのリンクを行います。

3.4.3 メインプログラムと演算ルーチンとのリンク

L80により、両方のオブジェクトをリンクします。その実行例を次に示します。

この2つをリンクして、この名前の"COM"ファイルを生成して、CP/Mへ戻る。

```

A>L80 HIGHLANG,MACHINE,LINKPROG/E/N/
Link-80  3.43  14-Apr-81  Copyright (c) 1981 Microsoft
Data      0103      281B      <1000B>

19227 Bytes Free
[011A  281B      40]

A> リンク終了。

```

Figure-3.4.5 メインプログラムと演算ルーチンとのリンク。

2つの"REL"ファイル、"HIGHLANG"と"MACHINE"をリンクして、1本の実行可能な"COM"ファイル "LINKPROG.COM" が生成されたはずですが、STATコマンドで確認してみましょう。

A>STAT LINKPROG.*! L80により生成された"COM"ファイルの確認。

RECS	BYTES	EXT	ACC
79	10K	1 R/W	A:LINKPROG.COM 簡単なプログラムであるが、10Kバイトにもなっている。

BYTES REMAINING ON A: 42K

A>

Figure-3.4.6 リンクにより生成された実行可能ファイルの確認。

このように、10Kバイトの"LINKPROG.COM" が生成されています。小さなプログラムであるのに、10Kバイトと、オブジェクト効率が良くないのは、コンパイル時にスイッチ"/O"を付けて、すべてのランタイム・ライブラリを取り込んだためであり、このランタイム・ライブラリを別ファイルのままにしておけば、オブジェクトはもっと小さくなります（バージョン5.30以上の場合）。

ではでき上がったプログラムを実行してみましょう。実行例を次に示します。

A>LINKPRG /完成したプログラムの実行.

INPUT PARAMETER 1 AND 2, (parm1,parm2) >? 250,750 /

250 + 750 = 1000

INPUT PARAMETER 1 AND 2, (parm1,parm2) >? 8080,6809 /

8080 + 6809 = 14889

INPUT PARAMETER 1 AND 2, (parm1,parm2) >? 8086,-86 /マイナスの値の例.

8086 + -86 = 8000

INPUT PARAMETER 1 AND 2, (parm1,parm2) >? ^CCtrl-Cの入力でCP/Mへ戻る.

STOP at address 0150BASCOSシステムからの出力.

A>

Figure-3.4.7 できたプログラムの実行.

このように合計が求められています.

注) 演算は16ビットの整数で取り扱われますので、-32767~+32767の範囲の数でなければなりません.

ここでの例題は非常に単純なものですが、この手法の応用は、複雑で時間のかかる計算や、高速性を必要とする機器の制御などのソフトウェアの開発に威力を発揮するでしょう.

4章 他の8bit CPUおよび 16bit CPUのソフト開発



8bitのCP/Mマシン上で、8080, 8085, Z80, 6502, 6800, 6809などの8bit CPU, および8086, 8088, Z8000などの16bit CPUのアセンブリ言語をアセンブルすることができます。これらは“クロス・アセンブラ”と呼ばれており、1台のCP/Mマシンで、多くの異なったCPUのソフト開発が可能となる便利なものです。クロス・アセンブラであっても、その機能は自分自身のCPU上で使用するアセンブラのどれをも上回る製品もあり、正にCP/Mならではの現象と言えます。

一方、8bitのCPUのアセンブリ言語のプログラムを、そのまま16bit CPUのアセンブリ言語のプログラムに、自動的に書き変えてしまう“トランスレータ”と呼ばれるソフトがあります。

この種のソフトは現時点では、8080ソース・コードを8086ソース・コードにトランスレートするものと、Z80ソース・コードを8086ソース・コードにトランスレートするものが発売されていますが、その他のCPUのものも近々には出現するのではないかと思います。

本章では、このような開発用ソフトの中から、6809アセンブラと8080→8086トランスレータを取り上げ、その使用例を紹介します。

4.1 ACT69の使用例

ソーシム社のクロス・アセンブラ、“ACT”シリーズには、現時点で、

ACT80 (インテル8080, 8085, Z80)

ACT65 (モステック6502)

ACT68 (モトローラ6800)

ACT86 (インテル8086, 8088)

ACT69 (モトローラ6809)

がそろっています。これらは、どれもほぼ同様なアセンブル機能を持っており、基本的な機能は3章で紹介した“MAC”や“MACRO-80”と大体同じマクロ・アセンブラですが、ACT独自のユニークな機能もいくつか追加されています。

では、ACT69の実行例を示しましょう。

ソース・ファイルは、ACT69のディスケットに含まれるテスト用のソース・ファイルを使用します。このソース・ファイルは、6809のすべてのオペレーション・コードとレジスタの組み合わせが、アセンブリ・ソース・プログラムとして書かれているものです。

このプログラムの冒頭には、“LINK”疑似命令でディスク上の別のファイル“ADRM6809.A69”をリンクするように記述されており、このリンクの様子も見ることができます。

ソース・ファイル“SMPL6809.A69”(ソース・ファイルのエクステンションは、“A69”でなければならない)をタイプアウトして次に示します。


```

A>TYPE SMPL6809.A69 .....6809のソース・ファイルのタイプアウト、
                           ファイル名のエクステンションは"A69"でなければならない。
TITLE 'Smpl6809 - MC6809 Opcode Listing.
Smpl6809 6809 Opcode Listing.

;
;
; Since there are so many addressing mode combinations,
; the first part of this list only has enough instructions
; to generate every possible first byte.
;
; The file ADRM6809.A69 has one of each address mode.

nn: equ 5
mmm: equ $FFEE

Link ADRM6809 ;all addressing modes .....ファイル"ADRM6809.A69"
                                         を呼び出し、当プログラムの
                                         最後にリンクする。

ORG $100

ABX ;3A

ADCA #nn ;89 05
ADCA nn ;99 05
ADCA nn,X ;A9 05
ADCA mmm ;B9 mmm

ADCB #nn ;C9 05
ADCB nn ;D9 05
ADCB B,X ;E9 85
ADCB mmm ;F9 mmm

ADDA #nn ;8B 05
ADDA nn ;9B 05
ADDA [B,X] ;AB 95
ADDA mmm ;BB FFEE
.
.

ORB #nn ;CA
ORB nn
ORB 0,-Y
ORB mmm

ORCC nn ;1A

PSHS PC,U,Y,X,DP,B,A,CC
PSHU PC,S,Y,X,DP,B,A,CC
PULS PC,U,Y,X,DP,B,A,CC
PULU PC,S,Y,X,DP,B,A,CC

ROL nn ;09 05
ROL ,X+ ;69 80
ROL mmm ;79 FFEE
ROLA ;49
ROLB ;59
.
.

TST nn ;0D
TST 0,-Y
TST mmm
TSTA

```



```

TSTB
;      Endx      DocOp69.asm
A>

```

Figure-4.1.1 6809のすべてのオペレーション・コードが含まれるテスト用ソース・ファイルのタイプアウト。

次に、プログラム中の疑似命令“LINK”で呼び出されるディスク上のもう1つのソース・ファイル“ADRM6809.A69”をタイプアウトして示します。これは、6809のすべてのアドレッシング・モードが記述されたテスト用のソース・ファイルです。

A>TYPE ADRM6809.A69/ディスク上のリンクされるもう1つのソース・ファイルのタイプアウト。

```

;      TITLE      'All 6809 Addressing Modes.'
;      Example of all 6809 Addressing Modes.

n5:    EQU        14
n8:    EQU        127
mm16:  EQU        0FEDCh

;      Accumulator Offset.

LDD     A,X
LDD     A,Y
LDD     A,S
LDD     A,U
LDD     B,X
LDD     B,Y
LDD     B,S
LDD     B,U
LDA     D,X
LDA     D,Y
LDA     D,S
LDA     D,U

LDD     [A,X]
LDD     [A,Y]
LDD     [A,S]
.
.

LDD     [n8,S]
LDD     [n8,U]
LDD     [n8,PC]
LDD     [mm16,X]
LDD     [mm16,Y]
LDD     [mm16,S]
LDD     [mm16,U]
LDD     [mm16,PC]

END
A>

```

Figure-4.1.2 プログラム中の疑似命令“LINK”で呼び出されるもう1つのソース・ファイルのタイプアウト。

ではACT69を起動して、アセンブラを実行してみましょう。その実行例を次に示します。

PRNファイルはドライブA：上に（HEXファイルは指定していないのでログイン・ディスク上に）。

ソース・ファイル名 リファレンス・マップはフル表示を行う。

A>ACT69 SMPL6809 L=A: R=F J ...ACT69によるアセンブル例。

SORCIM 6809 Assembler ver 3.5F
Pass 1 - Reading A:SMPL6809.A69
Pass 1 - Reading A:ADRM6809.A69
Pass 2 - Reading A:SMPL6809.A69
Pass 2 - Reading A:ADRM6809.A69
no ERRORS, 7 Labels, 6504h bytes not used. Program LWA = 0548h.

A> アセンブル終了。

Figure-4.1.3 ACT69によるアセンブラの実行。

このように、それぞれのパスで読み込まれるソース・ファイル名が表示されて行き、アセンブルが終了しました。

生成されたファイルをDIRで確認してみましょう。

A>DIR SMPL6809.*J ...アセンブルにより生成された各ファイルの確認。

A: SMPL6809 A69 : SMPL6809 H69 : SMPL6809 PRN
ソース・ファイル HEXファイル PRNファイル
A>

Figure-4.1.4 アセンブルにより生成されたファイルの確認。

この中で“H69”というエクステンションのファイルが、インテルHEX形式のファイルです。

次に、アセンブルの内容を見るために、PRNファイルをタイプアウトして、その中の数ページを示します。各ページには、ページ・ヘッダが付き1ページのライン数はデフォルトで61ラインとなっています。

以上紹介したのはACT19の、ごく基本的な実行例にすぎないことをお断りしておきます。

<pre> SDRCIN 6809 Assembler ver 3.5F 07/17/82 14:20 Page 1 Smp1409 - HC409 Opcode Listing. OpCode Since there are so many addressing mode combinations, the first part of this list only has enough instructions to generate every possible first byte. The file ASIM6809.M69 has one of each address mode. Link AddrMode9 all addressing modes 0000 0100 ORG #100 0100 3A ABX #A 0101 8905 ADCR Rnn #100 05 0102 9005 ADCR Rnn #100 05 0105 8905 ADCR Rnn #100 05 0107 89FE ADCR Rnn #FE 010A C905 ADCR Rnn #100 05 010C 8905 ADCR Rnn #100 05 010E 8905 ADCR Rnn #100 05 0110 89FE ADCR Rnn #FE 0113 8905 ADCR Rnn #100 05 0115 8905 ADCR Rnn #100 05 0117 8905 ADCR Rnn #100 05 0119 89FE ADCR Rnn #FE 011C C805 ADCR Rnn #100 05 011E 8905 ADCR Rnn #100 05 0120 8905 ADCR Rnn #100 05 0122 89FE ADCR Rnn #FE 0125 C3FFE ADCR Rnn #FE 0128 8905 ADCR Rnn #100 05 012A E3A2 ADCR Rnn #A2 012C E3FE ADCR Rnn #FE 012F F3FFE ADCR Rnn #FE 0132 8405 ANDA Rnn #100 0134 8405 ANDA Rnn #100 0136 84FE ANDA Rnn #FE 0139 84FE ANDA Rnn #FE 013C C405 ANDA Rnn #100 013E 8405 ANDA Rnn #100 0140 E4FE ANDA Rnn #FE 0143 84FE ANDA Rnn #FE 0146 1C05 ANDC Rnn #100 0148 0805 ASL Rnn #100 014A 08A2 ASL Rnn #A2 014C 08FE ASL Rnn #FE 014E 78FE ASL Rnn #FE 0151 4B ASL Rnn #100 </pre>	<pre> SDRCIN 6809 Assembler ver 3.5F 07/17/82 14:20 Page 9 All 6809 Addressing Modes. Example of all 6809 Addressing Modes. Accumulator Offset. 043C E0A6 LDD A, #1 043E E0A6 LDD A, #1 0440 E0A6 LDD A, #1 0442 E0A6 LDD A, #1 0444 E0A6 LDD A, #1 0446 E0A6 LDD A, #1 0448 E0A6 LDD A, #1 044A E0A6 LDD A, #1 044C E0A6 LDD A, #1 044E E0A6 LDD A, #1 0450 E0A6 LDD A, #1 0452 E0A6 LDD A, #1 0454 E0A6 LDD A, #1 0456 E0A6 LDD A, #1 0458 E0A6 LDD A, #1 045A E0A6 LDD A, #1 045C E0A6 LDD A, #1 045E E0A6 LDD A, #1 0460 E0A6 LDD A, #1 0462 E0A6 LDD A, #1 0464 E0A6 LDD A, #1 0466 E0A6 LDD A, #1 0468 E0A6 LDD A, #1 046A E0A6 LDD A, #1 046C E0A6 LDD A, #1 046E E0A6 LDD A, #1 0470 E0A6 LDD A, #1 0472 E0A6 LDD A, #1 0474 E0A6 LDD A, #1 0476 E0A6 LDD A, #1 0478 E0A6 LDD A, #1 047A E0A6 LDD A, #1 047C E0A6 LDD A, #1 047E E0A6 LDD A, #1 0480 E0A6 LDD A, #1 0482 E0A6 LDD A, #1 0484 E0A6 LDD A, #1 0486 E0A6 LDD A, #1 0488 E0A6 LDD A, #1 048A E0A6 LDD A, #1 048C E0A6 LDD A, #1 048E E0A6 LDD A, #1 0490 E0A6 LDD A, #1 0492 E0A6 LDD A, #1 0494 E0A6 LDD A, #1 0496 E0A6 LDD A, #1 0498 E0A6 LDD A, #1 049A E0A6 LDD A, #1 049C E0A6 LDD A, #1 049E E0A6 LDD A, #1 04A0 E0A6 LDD A, #1 04A2 E0A6 LDD A, #1 04A4 E0A6 LDD A, #1 04A6 E0A6 LDD A, #1 04A8 E0A6 LDD A, #1 04AA E0A6 LDD A, #1 04AC E0A6 LDD A, #1 04AE E0A6 LDD A, #1 04B0 E0A6 LDD A, #1 04B2 E0A6 LDD A, #1 04B4 E0A6 LDD A, #1 04B6 E0A6 LDD A, #1 04B8 E0A6 LDD A, #1 04BA E0A6 LDD A, #1 04BC E0A6 LDD A, #1 04BE E0A6 LDD A, #1 04C0 E0A6 LDD A, #1 04C2 E0A6 LDD A, #1 04C4 E0A6 LDD A, #1 04C6 E0A6 LDD A, #1 04C8 E0A6 LDD A, #1 04CA E0A6 LDD A, #1 04CC E0A6 LDD A, #1 04CE E0A6 LDD A, #1 04D0 E0A6 LDD A, #1 04D2 E0A6 LDD A, #1 04D4 E0A6 LDD A, #1 04D6 E0A6 LDD A, #1 04D8 E0A6 LDD A, #1 04DA E0A6 LDD A, #1 04DC E0A6 LDD A, #1 04DE E0A6 LDD A, #1 04E0 E0A6 LDD A, #1 04E2 E0A6 LDD A, #1 04E4 E0A6 LDD A, #1 04E6 E0A6 LDD A, #1 04E8 E0A6 LDD A, #1 04EA E0A6 LDD A, #1 04EC E0A6 LDD A, #1 04EE E0A6 LDD A, #1 04F0 E0A6 LDD A, #1 04F2 E0A6 LDD A, #1 04F4 E0A6 LDD A, #1 04F6 E0A6 LDD A, #1 04F8 E0A6 LDD A, #1 04FA E0A6 LDD A, #1 04FC E0A6 LDD A, #1 04FE E0A6 LDD A, #1 </pre>
<pre> SDRCIN 6809 Assembler ver 3.5F 07/17/82 14:20 Page 8 Smp1409 - HC409 Opcode Listing. OpCode 033B 0405 LBR Rnn #100 033D 0405 LBR Rnn #100 033F 74FE LBR Rnn #FE 0342 44 LBR Rnn #100 0343 94 LBR Rnn #100 0344 30 LBR Rnn #100 0345 0005 NES Rnn #100 0347 80E2 NES Rnn #E2 0349 78FE NES Rnn #FE 034B 80 NES Rnn #100 034E 12 NOP 034F 8905 ORA Rnn #100 0351 8905 ORA Rnn #100 0353 89FE ORA Rnn #FE 0356 0805 ORA Rnn #100 0358 0805 ORA Rnn #100 035A 08FE ORA Rnn #FE 035D 0805 ORA Rnn #100 035F 0805 ORA Rnn #100 0361 1A05 ORC Rnn #100 0363 3AFF RMB PC, U, V, W, X, Y, Z, CC 0365 3AFF RMB PC, U, V, W, X, Y, Z, CC 0367 3AFF RMB PC, U, V, W, X, Y, Z, CC 0369 3AFF RMB PC, U, V, W, X, Y, Z, CC 036B 0905 ROL Rnn #100 036D 0905 ROL Rnn #100 036F 78FE ROL Rnn #FE 0371 09 ROL Rnn #100 0373 09 ROL Rnn #100 0375 09 ROL Rnn #100 0377 09 ROL Rnn #100 0379 09 ROL Rnn #100 037B 09 ROL Rnn #100 037D 09 ROL Rnn #100 037F 09 ROL Rnn #100 0381 1A05 ORC Rnn #100 0383 3AFF RMB PC, U, V, W, X, Y, Z, CC 0385 3AFF RMB PC, U, V, W, X, Y, Z, CC 0387 3AFF RMB PC, U, V, W, X, Y, Z, CC 0389 3AFF RMB PC, U, V, W, X, Y, Z, CC 038B 0905 ROL Rnn #100 038D 0905 ROL Rnn #100 038F 78FE ROL Rnn #FE 0391 09 ROL Rnn #100 0393 09 ROL Rnn #100 0395 09 ROL Rnn #100 0397 09 ROL Rnn #100 0399 09 ROL Rnn #100 039B 09 ROL Rnn #100 039D 09 ROL Rnn #100 039F 09 ROL Rnn #100 03A1 0905 ROL Rnn #100 03A3 0905 ROL Rnn #100 03A5 0905 ROL Rnn #100 03A7 0905 ROL Rnn #100 03A9 0905 ROL Rnn #100 03AB 0905 ROL Rnn #100 03AD 0905 ROL Rnn #100 03AF 0905 ROL Rnn #100 03B1 0905 ROL Rnn #100 03B3 0905 ROL Rnn #100 03B5 0905 ROL Rnn #100 03B7 0905 ROL Rnn #100 03B9 0905 ROL Rnn #100 03BB 0905 ROL Rnn #100 03BD 0905 ROL Rnn #100 03BF 0905 ROL Rnn #100 03C1 0905 ROL Rnn #100 03C3 0905 ROL Rnn #100 03C5 0905 ROL Rnn #100 03C7 0905 ROL Rnn #100 03C9 0905 ROL Rnn #100 03CB 0905 ROL Rnn #100 03CD 0905 ROL Rnn #100 03CF 0905 ROL Rnn #100 03D1 0905 ROL Rnn #100 03D3 0905 ROL Rnn #100 03D5 0905 ROL Rnn #100 03D7 0905 ROL Rnn #100 03D9 0905 ROL Rnn #100 03DB 0905 ROL Rnn #100 03DD 0905 ROL Rnn #100 03DF 0905 ROL Rnn #100 03E1 0905 ROL Rnn #100 03E3 0905 ROL Rnn #100 03E5 0905 ROL Rnn #100 03E7 0905 ROL Rnn #100 03E9 0905 ROL Rnn #100 03EB 0905 ROL Rnn #100 03ED 0905 ROL Rnn #100 03EF 0905 ROL Rnn #100 03F1 0905 ROL Rnn #100 03F3 0905 ROL Rnn #100 03F5 0905 ROL Rnn #100 03F7 0905 ROL Rnn #100 03F9 0905 ROL Rnn #100 03FB 0905 ROL Rnn #100 03FD 0905 ROL Rnn #100 03FF 0905 ROL Rnn #100 </pre>	<pre> SDRCIN 6809 Assembler ver 3.5F 07/17/82 14:20 Page 11 All 6809 Addressing Modes. Example of all 6809 Addressing Modes. Accumulator Offset. 052A E0FE LDD A, #FE 052C E0FE LDD A, #FE 052E E0FE LDD A, #FE 0530 E0FE LDD A, #FE 0532 E0FE LDD A, #FE 0534 E0FE LDD A, #FE 0536 E0FE LDD A, #FE 0538 E0FE LDD A, #FE 053A E0FE LDD A, #FE 053C E0FE LDD A, #FE 053E E0FE LDD A, #FE 0540 E0FE LDD A, #FE 0542 E0FE LDD A, #FE 0544 E0FE LDD A, #FE 0546 E0FE LDD A, #FE 0548 E0FE LDD A, #FE 054A E0FE LDD A, #FE 054C E0FE LDD A, #FE 054E E0FE LDD A, #FE 0550 E0FE LDD A, #FE 0552 E0FE LDD A, #FE 0554 E0FE LDD A, #FE 0556 E0FE LDD A, #FE 0558 E0FE LDD A, #FE 055A E0FE LDD A, #FE 055C E0FE LDD A, #FE 055E E0FE LDD A, #FE 0560 E0FE LDD A, #FE 0562 E0FE LDD A, #FE 0564 E0FE LDD A, #FE 0566 E0FE LDD A, #FE 0568 E0FE LDD A, #FE 056A E0FE LDD A, #FE 056C E0FE LDD A, #FE 056E E0FE LDD A, #FE 0570 E0FE LDD A, #FE 0572 E0FE LDD A, #FE 0574 E0FE LDD A, #FE 0576 E0FE LDD A, #FE 0578 E0FE LDD A, #FE 057A E0FE LDD A, #FE 057C E0FE LDD A, #FE 057E E0FE LDD A, #FE 0580 E0FE LDD A, #FE 0582 E0FE LDD A, #FE 0584 E0FE LDD A, #FE 0586 E0FE LDD A, #FE 0588 E0FE LDD A, #FE 058A E0FE LDD A, #FE 058C E0FE LDD A, #FE 058E E0FE LDD A, #FE 0590 E0FE LDD A, #FE 0592 E0FE LDD A, #FE 0594 E0FE LDD A, #FE 0596 E0FE LDD A, #FE 0598 E0FE LDD A, #FE 059A E0FE LDD A, #FE 059C E0FE LDD A, #FE 059E E0FE LDD A, #FE 05A0 E0FE LDD A, #FE 05A2 E0FE LDD A, #FE 05A4 E0FE LDD A, #FE 05A6 E0FE LDD A, #FE 05A8 E0FE LDD A, #FE 05AA E0FE LDD A, #FE 05AC E0FE LDD A, #FE 05AE E0FE LDD A, #FE 05B0 E0FE LDD A, #FE 05B2 E0FE LDD A, #FE 05B4 E0FE LDD A, #FE 05B6 E0FE LDD A, #FE 05B8 E0FE LDD A, #FE 05BA E0FE LDD A, #FE 05BC E0FE LDD A, #FE 05BE E0FE LDD A, #FE 05C0 E0FE LDD A, #FE 05C2 E0FE LDD A, #FE 05C4 E0FE LDD A, #FE 05C6 E0FE LDD A, #FE 05C8 E0FE LDD A, #FE 05CA E0FE LDD A, #FE 05CC E0FE LDD A, #FE 05CE E0FE LDD A, #FE 05D0 E0FE LDD A, #FE 05D2 E0FE LDD A, #FE 05D4 E0FE LDD A, #FE 05D6 E0FE LDD A, #FE 05D8 E0FE LDD A, #FE 05DA E0FE LDD A, #FE 05DC E0FE LDD A, #FE 05DE E0FE LDD A, #FE 05E0 E0FE LDD A, #FE 05E2 E0FE LDD A, #FE 05E4 E0FE LDD A, #FE 05E6 E0FE LDD A, #FE 05E8 E0FE LDD A, #FE 05EA E0FE LDD A, #FE 05EC E0FE LDD A, #FE 05EE E0FE LDD A, #FE 05F0 E0FE LDD A, #FE 05F2 E0FE LDD A, #FE 05F4 E0FE LDD A, #FE 05F6 E0FE LDD A, #FE 05F8 E0FE LDD A, #FE 05FA E0FE LDD A, #FE 05FC E0FE LDD A, #FE 05FE E0FE LDD A, #FE </pre>
<pre> SDRCIN 6809 Assembler ver 3.5F 07/17/82 14:20 Page 9 Smp1409 - HC409 Opcode Listing. OpCode 0423 1F50 TFR PC, D 0425 1F50 TFR PC, D 0427 1F50 TFR PC, D 0429 1F50 TFR PC, D 042B 1F50 TFR PC, D 042D 1F50 TFR PC, D 042F 1F50 TFR PC, D 0431 1F50 TFR PC, D 0433 1F50 TFR PC, D 0435 1F50 TFR PC, D 0437 1F50 TFR PC, D 0439 1F50 TFR PC, D 043B 1F50 TFR PC, D 043D 1F50 TFR PC, D 043F 1F50 TFR PC, D 0441 1F50 TFR PC, D 0443 1F50 TFR PC, D 0445 1F50 TFR PC, D 0447 1F50 TFR PC, D 0449 1F50 TFR PC, D 044B 1F50 TFR PC, D 044D 1F50 TFR PC, D 044F 1F50 TFR PC, D 0451 1F50 TFR PC, D 0453 1F50 TFR PC, D 0455 1F50 TFR PC, D 0457 1F50 TFR PC, D 0459 1F50 TFR PC, D 045B 1F50 TFR PC, D 045D 1F50 TFR PC, D 045F 1F50 TFR PC, D 0461 1F50 TFR PC, D 0463 1F50 TFR PC, D 0465 1F50 TFR PC, D 0467 1F50 TFR PC, D 0469 1F50 TFR PC, D 046B 1F50 TFR PC, D 046D 1F50 TFR PC, D 046F 1F50 TFR PC, D 0471 1F50 TFR PC, D 0473 1F50 TFR PC, D 0475 1F50 TFR PC, D 0477 1F50 TFR PC, D 0479 1F50 TFR PC, D 047B 1F50 TFR PC, D 047D 1F50 TFR PC, D 047F 1F50 TFR PC, D 0481 1F50 TFR PC, D 0483 1F50 TFR PC, D 0485 1F50 TFR PC, D 0487 1F50 TFR PC, D 0489 1F50 TFR PC, D 048B 1F50 TFR PC, D 048D 1F50 TFR PC, D 048F 1F50 TFR PC, D 0491 1F50 TFR PC, D 0493 1F50 TFR PC, D 0495 1F50 TFR PC, D 0497 1F50 TFR PC, D 0499 1F50 TFR PC, D 049B 1F50 TFR PC, D 049D 1F50 TFR PC, D 049F 1F50 TFR PC, D 04A1 1F50 TFR PC, D 04A3 1F50 TFR PC, D 04A5 1F50 TFR PC, D 04A7 1F50 TFR PC, D 04A9 1F50 TFR PC, D 04AB 1F50 TFR PC, D 04AD 1F50 TFR PC, D 04AF 1F50 TFR PC, D 04B1 1F50 TFR PC, D 04B3 1F50 TFR PC, D 04B5 1F50 TFR PC, D 04B7 1F50 TFR PC, D 04B9 1F50 TFR PC, D 04BB 1F50 TFR PC, D 04BD 1F50 TFR PC, D 04BF 1F50 TFR PC, D 04C1 1F50 TFR PC, D 04C3 1F50 TFR PC, D 04C5 1F50 TFR PC, D 04C7 1F50 TFR PC, D 04C9 1F50 TFR PC, D 04CB 1F50 TFR PC, D 04CD 1F50 TFR PC, D 04CF 1F50 TFR PC, D 04D1 1F50 TFR PC, D 04D3 1F50 TFR PC, D 04D5 1F50 TFR PC, D 04D7 1F50 TFR PC, D 04D9 1F50 TFR PC, D 04DB 1F50 TFR PC, D 04DD 1F50 TFR PC, D 04DF 1F50 TFR PC, D 04E1 1F50 TFR PC, D 04E3 1F50 TFR PC, D 04E5 1F50 TFR PC, D 04E7 1F50 TFR PC, D 04E9 1F50 TFR PC, D 04EB 1F50 TFR PC, D 04ED 1F50 TFR PC, D 04EF 1F50 TFR PC, D 04F1 1F50 TFR PC, D 04F3 1F50 TFR PC, D 04F5 1F50 TFR PC, D 04F7 1F50 TFR PC, D 04F9 1F50 TFR PC, D 04FB 1F50 TFR PC, D 04FD 1F50 TFR PC, D 04FF 1F50 TFR PC, D </pre>	<pre> SDRCIN 6809 Assembler ver 3.5F 07/17/82 14:20 Page 10 Smp1409 - HC409 Opcode Listing. OpCode 0423 1F50 TFR PC, D 0425 1F50 TFR PC, D 0427 1F50 TFR PC, D 0429 1F50 TFR PC, D 042B 1F50 TFR PC, D 042D 1F50 TFR PC, D 042F 1F50 TFR PC, D 0431 1F50 TFR PC, D 0433 1F50 TFR PC, D 0435 1F50 TFR PC, D 0437 1F50 TFR PC, D 0439 1F50 TFR PC, D 043B 1F50 TFR PC, D 043D 1F50 TFR PC, D 043F 1F50 TFR PC, D 0441 1F50 TFR PC, D 0443 1F50 TFR PC, D 0445 1F50 TFR PC, D 0447 1F50 TFR PC, D 0449 1F50 TFR PC, D 044B 1F50 TFR PC, D 044D 1F50 TFR PC, D 044F 1F50 TFR PC, D 0451 1F50 TFR PC, D 0453 1F50 TFR PC, D 0455 1F50 TFR PC, D 0457 1F50 TFR PC, D 0459 1F50 TFR PC, D 045B 1F50 TFR PC, D 045D 1F50 TFR PC, D 045F 1F50 TFR PC, D 0461 1F50 TFR PC, D 0463 1F50 TFR PC, D 0465 1F50 TFR PC, D 0467 1F50 TFR PC, D 0469 1F50 TFR PC, D 046B 1F50 TFR PC, D 046D 1F50 TFR PC, D 046F 1F50 TFR PC, D 0471 1F50 TFR PC, D 0473 1F50 TFR PC, D 0475 1F50 TFR PC, D 0477 1F50 TFR PC, D 0479 1F50 TFR PC, D 047B 1F50 TFR PC, D 047D 1F50 TFR PC, D 047F 1F50 TFR PC, D 0481 1F50 TFR PC, D 0483 1F50 TFR PC, D 0485 1F50 TFR PC, D 0487 1F50 TFR PC, D 0489 1F50 TFR PC, D 048B 1F50 TFR PC, D 048D 1F50 TFR PC, D 048F 1F50 TFR PC, D 0491 1F50 TFR PC, D 0493 1F50 TFR PC, D 0495 1F50 TFR PC, D 0497 1F50 TFR PC, D 0499 1F50 TFR PC, D 049B 1F50 TFR PC, D 049D 1F50 TFR PC, D 049F 1F50 TFR PC, D 04A1 1F50 TFR PC, D 04A3 1F50 TFR PC, D 04A5 1F50 TFR PC, D 04A7 1F50 TFR PC, D 04A9 1F50 TFR PC, D 04AB 1F50 TFR PC, D 04AD 1F50 TFR PC, D 04AF 1F50 TFR PC, D 04B1 1F50 TFR PC, D 04B3 1F50 TFR PC, D 04B5 1F50 TFR PC, D 04B7 1F50 TFR PC, D 04B9 1F50 TFR PC, D 04BB 1F50 TFR PC, D 04BD 1F50 TFR PC, D 04BF 1F50 TFR PC, D 04C1 1F50 TFR PC, D 04C3 1F50 TFR PC, D 04C5 1F50 TFR PC, D 04C7 1F50 TFR PC, D 04C9 1F50 TFR PC, D 04CB 1F50 TFR PC, D 04CD 1F50 TFR PC, D 04CF 1F50 TFR PC, D 04D1 1F50 TFR PC, D 04D3 1F50 TFR PC, D 04D5 1F50 TFR PC, D 04D7 1F50 TFR PC, D 04D9 1F50 TFR PC, D 04DB 1F50 TFR PC, D 04DD 1F50 TFR PC, D 04DF 1F50 TFR PC, D 04E1 1F50 TFR PC, D 04E3 1F50 TFR PC, D 04E5 1F50 TFR PC, D 04E7 1F50 TFR PC, D 04E9 1F50 TFR PC, D 04EB 1F50 TFR PC, D 04ED 1F50 TFR PC, D 04EF 1F50 TFR PC, D 04F1 1F50 TFR PC, D 04F3 1F50 TFR PC, D 04F5 1F50 TFR PC, D 04F7 1F50 TFR PC, D 04F9 1F50 TFR PC, D 04FB 1F50 TFR PC, D 04FD 1F50 TFR PC, D 04FF 1F50 TFR PC, D </pre>

Figure-4.1.5 ACT69により作成されたPRNリスト

4.2 XLT86の使用例

4.2.1 動作の概略

XLT86は、8bitのCP/Mマシン上で、8080のアセンブリ言語プログラムを、16bit CPUである8086用のプログラムにトランスレートするものであり、今までに8080用として作られたプログラムを、人手による書き替えを必要とせず、8086用プログラムに変換することができるものです。

この機能を図示したものを次に示します。

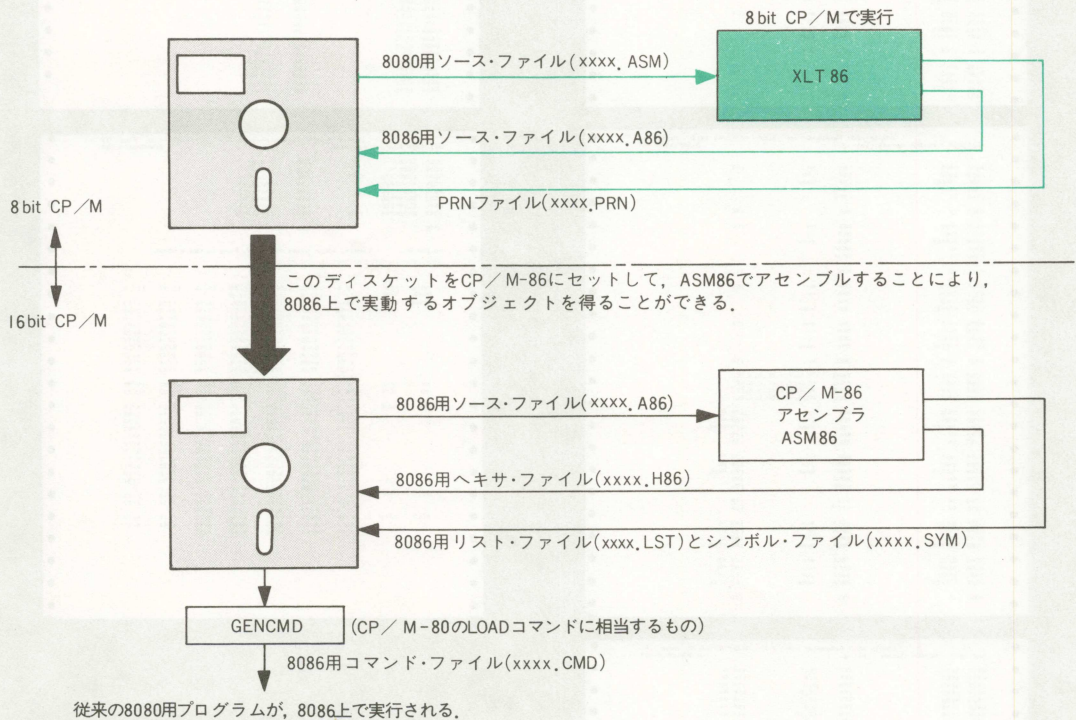


Figure-4.2.1 8086トランスレータ：XLT86の動作原理

この図からもわかるように、まず結論から言えば、8080用に開発されたプログラムを、そのままの機能を持った8086用のプログラムに自動的に変換することができるということです。

では、Figure-4.2.1を順を追って説明しましょう。

まず、8080用に開発されたプログラムがあり、そのアセンブリ・ソース・ファイルがディスク上にあります。例えば、標準的なCP/Mのシステム・ディスクに必ず含まれている、ダンプ・プログラムの“DUMP.COM”と、そのソース・ファイル“DUMP.ASM”などを想定すれば良いでしょう。“DUMP.COM”はもちろんすべてのCP/M上で実行することができますが、16bitの8086上では当然ながら動作しません。

8080用に作ったプログラムが、そのまま8086用としても使えれば、こんなありがたいことはないのですが、これを可能にするのがXLT86である訳です。

ダンプ・プログラムを例にとると“COM”ファイルの“DUMP.COM”は、16bitに関しては全く何の役にも立たないので、これは無視します。問題は、そのアセンブリ・ソース・ファイル“DUMP.ASM”にあります。

XLT86は、このアセンブリ・ソース・ファイルを取り込み、そのプログラムのグローバルなデータの流れを分析し、適正な8086のインストラクションに置き替えます。XLT86の生命は、この“global data flow analysis”にあり、これが、同種のトランスレータに比べ、数10%も効率の良いインストラクションを発生させているものと思われます。

このように、8080のアセンブリ・ソース・ファイルからトランスレートされた8086用のアセンブリ・ソース・ファイルは、任意のドライブ上にファイル名“xxxx.A86”としてセーブされます。そして、このディスクを、CP/M-86マシンにセットし（ディスクのファイル構造は、CP/M-80、CP/M-86とも同じなので、そのままどちらをどちらへ持っていてもリード・ライト可能です）、そのアセンブラ“ASM86”でアセンブルし、GENCMD（CP/M-80のLOADコマンドに相当するもの）で、実行可能なコマンド・ファイルとすることにより、8086上で元の8080の場合と同様にダンプ・プログラムを実行することができます。

4.2.2 XLT86の実行例

サンプル・プログラムとしては、CP/Mユーザーであれば誰でも知っている、先程のダンプ・プログラムが適当なのですが、紙面の関係で、もっと短いプログラムである本書2.2章のファンクション：9での実習プログラムFigure-2.2.15を取り上げてみます。

そのアセンブリ・ソース・プログラムのファイル名を、ここでは“FUNC9.ASM”とし、そのタイプアウトを再度次に示しておきます。

このプログラムの内容は非常に簡単なもので、キー入力に応じてそれぞれのメッセージが表示されるだけのものです（2.2章のファンクション：9を参照）。


```

A>TYPE FUNC9.ASM/ .....サンプル・プログラムのアセンブリ・ソース・ファイルをタイプアウトする.

;*****
;  FUNCTION  9: PRINT STRING
;*****

START:    ORG      100H

          MVI      C,9
          LXI      D,MSGINP
          CALL     0005H

          MVI      C,1
          CALL     0005H

          CPI      'M'
          JZ       MONGOUT
          CPI      'N'
          JZ       NIGTOUT
          CPI      1AH ; =^Z
          RZ

          MVI      C,2
          MVI      E,'?'
          CALL     0005H

          JMP      START

MONGOUT:  MVI      C,9
          LXI      D,MSGMONG
          CALL     0005H
          JMP      START

NIGTOUT:  MVI      C,9
          LXI      D,MSGNIGT
          CALL     0005H
          JMP      START

MSGINP:   DB       0DH,0AH,'input M)orning or N)ight --->$'
MSGMONG:  DB       0DH,0AH,0AH,'GOOD MORNING! THIS IS FUNCTION 9.',0DH,0AH,'$'
MSGNIGT:  DB       0DH,0AH,0AH,'GOOD NIGHT! THIS IS FUNCTION 9.',0DH,0AH,'$'

          END

A>

```

Figure-4.2.2 8080のサンプル・プログラムのアセンブリ・ソース・ファイルのタイプアウト.

この8080用のソース・ファイル "FUNC9.ASM" に対し、XLT86を実行し、8086用のソース・ファイルが生成される過程を紹介します。

XLT86の実行例を次に示します。

コマンド・ラインの [] の中に指定した3つのパラメータは、

80……生成されるPRNファイル中に8080のアセンブリ・リストを作成する、

86……同じくPRNファイル中に8086用のラインとステートメントのリストを作成する、

B……同じくPRNファイル中にBASIC BLOCKのリストを作成する、

という意味を持っています。

3つのパラメータ、「80」「86」「B」を指定（本文参照）。

A>XLT86 FUNC9[B086B] /8080用ソース・プログラム“FUNC9.ASM”に対して、XLT86を実行。

XLT86 Version 1.1

Copyright (c) 1981 以下の5ステップが順次実行されて行く。

Digital Research

Symbol Setup8080ソース・プログラムの各シンボルのロケーション確認のためのステップ。

Setup Blocksデータの流れを分析するためのBASIC BLOCKを決定するステップ。

Join Blocks互いのBASIC BLOCKを結合して“Directed Graph”を構成するステップ。

List Blocksフロー分析によるBASIC BLOCKのオプションなリストを作成するステップ。

Translate-86フロー分析に従って、8080インストラクションを8086インストラクションに変換するステップ

A>トランスレート終了。

Figure-4.2.3 8080のソースプログラム “FUNC9. ASM” に対するXLT86の実行例。

このように、5つのステップを経てトランスレートの作業が終了しました。
生成された各ファイルをDIRで確認してみましょう。

A>DIR FUNC9.* /XLT86の実行によって生成される各ファイルの確認。

A:	FUNC9	ASM :	FUNC9	PRN :	FUNC9	A86
	8080ソース・プログラム		各種リストファイル		生成された8086用ソース・プログラム	

A>

Figure-4.2.4 XLT86の実行により生成された各ファイルの確認。

8086用のソース・ファイル “FUNC9. A86” が生成されています。8080用のソース・プログラムが、どのように8086用のソース・プログラムにトランスレートされているでしょう。“FUNC9. A86”をタイプアウトして次に示します。

A>TYPE FUNC9.A86/でき上った8086用ソース・ファイルのタイプアウト.

```

M      EQU      Byte Ptr 0[BX]
;*****
;      FUNCTION 9: PRINT STRING
;*****
      ORG      100H
START:
      MOV      CL,9
      MOV      DX,(Offset MSGINP)
      INT      224
      MOV      CL,1
      INT      224
      CMP      AL,'M'
      JZ       MONGOUT
      CMP      AL,'N'
      JZ       NIGTOUT
      CMP      AL,1AH          ;=^Z
      JNZ      L_1
      RET

L_1:
      MOV      CL,2
      MOV      DL,'?'
      INT      224
      JMP      START

MONGOUT:
      MOV      CL,9
      MOV      DX,(Offset MSGMONG)
      INT      224
      JMP      START

NIGTOUT:
      MOV      CL,9
      MOV      DX,(Offset MSGNIGT)
      INT      224
      JMP      START

L_2      EQU      $
      DSEG
      ORG      Offset L_2
MSGINP  DB      0DH,0AH,'input M)orning or N)ight --->$'
MSGMONG DB      0DH,0AH,0AH,'GOOD MORNING! THIS IS FUNCTION 9.',0DH,0AH,'$'
MSGNIGT DB      0DH,0AH,0AH,'GOOD NIGHT! THIS IS FUNCTION 9.',0DH,0AH,'$'
      END

```

A>

Figure-4.2.5 でき上った8086用ソース・ファイルのタイプアウト.

このソース・ファイルをCP/Mのアセンブラ "ASM86" でアセンブルすれば良い訳です。

次に、8080ソース・プログラムが、8086用にトランスレートされる過程の分析リストなどが含まれるPRNファイル"FUNC9.PRN"の中から、パラメータ"B"と"86"によって生成されたBASIC BLOCKのリストと、8086用のラインとステートメントのリストを示します。

ここで示されているそれぞれのBASIC BLOCKが、データ・フローの分析によって適切に組み合わせられ、8086用にトランスレートされていく訳です。


```

LIST OF BASIC BLOCKS

Block At 0005 (aubr, ABA = 0000)
  Entry Active: -----
  Exit Active: -----
  =====
  !stat#: opcode uses : op : v1 : v2 : opcode kills : live regs :
  =====
  |
  |
Block At 0100 (code), ABA = 0100
  Entry Active: B--HL-A--
  Exit Active: B--DEH-A--
  =====
  !stat#: opcode uses : op : v1 : v2 : opcode kills : live regs :
  =====
  | 7:-----
  |      |HW1| C1 0P1-C-----|B--HL-A--|
  | 8:-----
  |      |IR1| D013A1--DE-----|B--DEH-A--|
  | 9:-----
  |      |CALL(0005)|-----|B--DEH-A--|
  =====
Block At 0108 (code), ABA = 0108
  Entry Active: B--DEH-A--
  Exit Active: B--DEH-A--
  =====
  !stat#: opcode uses : op : v1 : v2 : opcode kills : live regs :
  =====
  | 11:-----
  |      |HW1| C1 011-C-----|B--DEH-A--|
  | 12:-----
  |      |CALL(0005)|-----|B--DEH-A--|
  =====
Block At 010D (code), ABA = 010D
  Entry Active: B--DEH-A--
  Exit Active: B--DEH-A--
  =====
  !stat#: opcode uses : op : v1 : v2 : opcode kills : live regs :
  =====
  | 14:-----
  |      |A14--COP1 A01-----|DZSP1 B--DEH-A--|
  | 15:-----
  |      |J21 1012A1-----|B--DEH-A--|
  =====
Block At 0112 (code), ABA = 0114
  Entry Active: B--DEH-A--
  Exit Active: B--DEH-A--
  =====
  !stat#: opcode uses : op : v1 : v2 : opcode kills : live regs :
  =====
  | 16:-----
  |      |A14--COP1 AE1-----|DZSP1 B--DEH-A--|
  | 17:-----
  |      |J21 1012F1-----|B--DEH-A--|
  =====
Block At 0117 (aubr), ABA = 011B
  Entry Active: B--HL-A--
  Exit Active: B--HL-A--
  =====
  !stat#: opcode uses : op : v1 : v2 : opcode kills : live regs :
  =====
  | 18:-----
  |      |A14--COP1 1A1-----|DZSP1 B--DEH-A--|
  | 19: B--DEH-A--ZDZSP1|-----|B--HL-A--|
  =====
Block At 011A (code), ABA = 0120
  Entry Active: B--HL-A--
  Exit Active: B--HL-A--
  =====
  !stat#: opcode uses : op : v1 : v2 : opcode kills : live regs :
  =====

```

```

Entry Active: BCDEH4-A0ZSP! Exit Active: BCDEH4-A0ZSP!
-----
!state! opcode uses : op ! v1 ! v2 ! opcode kills ! live regs !
-----

```

```

i 21-----IMWI | C| 02-C-----|B---HL-A---|
i 22-----IMWI | C| 3F---E-----|B---HL-A---|
i 23-----ICALL|0005|-----|B---HL-A---|

Block At 0121 (code), ABA = 0127
Entry Active: B---HL-A---
!start! opcode uses : op | v1 | v2 | opcode kills | live regs |
i 25-----JMPF |0100| |-----|B---HL-A---|

Block At 0124 (code), ABA = 012A
Entry Active: B---HL-A---
!start! opcode uses : op | v1 | v2 | opcode kills | live regs |
i 28-----LRI | C| 09-C-----|B---HL-A---|
i 29-----ILRI | D| 0105E--DE-----|B---HL-A---|
i 30-----ICALL|0005|-----|B---HL-A---|

Block At 012C (code), ABA = 0132
Entry Active: B---HL-A---
Exit Active: B---HL-A---
!start! opcode uses : op | v1 | v2 | opcode kills | live regs |
i 31-----JMPF |0100| |-----|B---HL-A---|

Block At 013F (code), ABA = 0135
Entry Active: B---HL-A---
Exit Active: B---HL-A---
!start! opcode uses : op | v1 | v2 | opcode kills | live regs |
i 34-----IMWI | C| 09-C-----|B---HL-A---|
i 35-----ILRI | D| 0105E--DE-----|B---HL-A---|
i 36-----ICALL|0005|-----|B---HL-A---|

Block At 0137 (code), ABA = 013D
Entry Active: B---HL-A---
Exit Active: B---HL-A---
!start! opcode uses : op | v1 | v2 | opcode kills | live regs |
i 37-----JMPF |0100| |-----|B---HL-A---|

Block At 013A (data), ABA = 0140
Entry Active: -----
Exit Active: -----
!start! opcode uses : op | v1 | v2 | opcode kills | live regs |
i 39-----IDB |00024|-----|-----|
i 40-----IDB |00029|-----|-----|
i 41-----IDB |00025|-----|-----|

Block At 01A6 (code), ABA = 01B0

```

```

1      0 W          EDI             Byte Ptr= C8B3
2      1 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
3      2 FUNCTION 9: PRINT STRING
4      3 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
5      4
6      5 ORG          100H
7      6 START:
8      7 MOV          CL, 9
9      8 MOV          DI, (Offset MSGINP)
10     9 INT         224
11    10 MOV          CX, 1
12    11 INT         224
13    12 INT         224
14    13 CMP          AL, 'N'
15    14 JZ           NODISOUT
16    15 MOV          J1, 7
17    16 INT         NIDISOUT
18    17 CMP          RC, 1AH
19    18 JNE          L_1
20    19 RET
21    20 L_1:
22    21 MOV          CL, 2
23    22 MOV          DI, ??
24    23 INT         224
25    24 JMPB        START
26    25
27    26 NODISOUT:
28    27 MOV          CL, 9
29    28 MOV          DI, (Offset MSGNIGHT)
30    29 INT         224
31    30 JMPB        START
32    31
33    32 NIDISOUT:
34    33 MOV          CL, 9
35    34 MOV          DI, (Offset MSGNIGHT)
36    35 INT         224
37    36 JMPB        START
38    37
39    38 L_2:
40    39 MOV          DBEB
41    40 MOV          DBEB
42    41 MSGINIST DB   ODI, ODI, Input Morning or Night ---->*,
43    42                                ODI, ODI, "GOOD MORNING! THIS IS FUNCTION 9.", ODI, ODI
44    43 MSGNIST DB   ODI, ODI, "GOOD NIGHT! THIS IS FUNCTION 9.", ODI, ODI

```

Figure-4.2.6 PRNファイルに含まれるBASIC BLOCKのリスト.

他の8bit CPUおよび16bit CPUのソフト開発

BASIC BLOCKのリスト中の (subr) はサブルーチン, (code) はメイン・ラインコード, (data) はデータ・ブロックを表します.

その他, XLT86は, 例えばCP/M-86の "コンパクト・メモリモデル" 用のソース・プログラムを作ったり, コードとデータ・セグメントをオーバーラップさせないようにするなど, いくつかの指示をXLT86の実行時にパラメータとして与えることができます.

5章 各種高級言語による 同一主題ソフト開発例



CP/M は、高級言語の分野においても、他の OS に比べ圧倒的に優位です。2～3の代表的な言語にとどまらず、いろいろな言語プロセッサを使いたい場合、どうしても CP/M に頼らざるを得ないのです。

例えば、代表的な言語である COBOL では、ANSI 標準 COBOL (ANSI X3.23 1974) のレベルIIが走り (CIS COBOL) これは、プロフェッショナルのプログラマが十分満足できる内容です (これには便利な "COMPUTE" ステートメントも使用できる)。

また、今まで限られた機種でしか利用できず、使いたくても使えなかった人が多い APL など、国内の販売店で入手できるようになりました。CP/M 上で走る APL を具体的に紹介するのは本書が最初と思いますが、APL が、CP/M マシン上で安価に誰もが利用できるということは、スカラー、ベクトル、マトリックスなどのデータ解析、数値解析、それにグラフィックスなどの分野の人達に大きな刺激を与えるものと思われます。

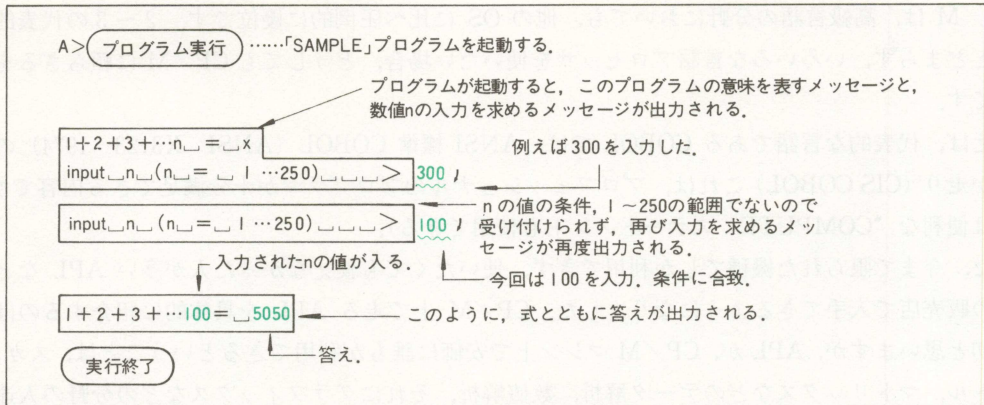
この章では、一般のユーザーが、容易に入手可能な言語を11種類取り上げました。そこで、簡単な問題を全部の言語に与え、その問題解決をそれぞれの言語で実現してみましょう。

取り上げた11種類の言語と、その商品とメーカー名を次に示します。購入先などは本書の巻末付録 C に示しておきます。

1. ^{コボル}COBOL……CIS COBOL (米 MICRO FOCUS 社) 中間言語コンパイラ
2. ^{フォートラン}FORTRAN……FORTRAN-80 (米 Microsoft 社) コンパイラ
3. ^{ベーシック}BASIC……CB-80 (米 Digital Research 社) コンパイラ
4. ^{パスカル}PASCAL……Pascal/MT+ (米 Digital Research 社) コンパイラ
5. ^{ピーエルワン}PL/I……PL/I-80 (米 Digital Research 社) コンパイラ
6. ^{ピーエルエム}PL/M……PLMX (米 Systems Consultants 社) コンパイラ
7. ^{シー}C……BDS C Compiler (米 BD Software 社) コンパイラ
8. ^{フォース}FORTH……Rgy FORTH (日 リギーコーポレーション) コンパイラ
9. ^{リスプ}LISP……muLISP (米 The SoftWarehouse 社) 中間言語コンパイラ/インタープリタ
10. ^{アルゴル}ALGOL……ALGOL-M (米 Mark Moranville) 中間言語コンパイラ/インタープリタ
11. ^{エービーエル}APL……SOFTRONICS APL\80 (米 SOFTRONICS 社) インタープリタ

以上の言語に対する課題は次のようなものです。

プログラム名は、「SAMPLE」とします。その内容は、プログラムを実行した場合の具体的なスクリーン表示例で示します。



つまり、ただの数列の総和（ $1 + 2 + 3 + \dots + n$ ）を求めるプログラムですが、その中には、コンソールとの入出力、演算、判断の要素が入っています（入力が 0 や、1、2、3 の場合、答えの式がおかしいなど、細かいことには目をつぶって下さい）。

では、この課題「SAMPLE」を、COBOL から順にそれぞれの言語で実現して行きましょう。

ここでは、これらの言語とその使い方についての詳細な解説を行うのが目的ではありません。今まで、一部のミニコンピュータや、大型コンピュータでしか使うことができなかった言語の、そのフルセットあるいはサブセットレベルのものが、CP/M マシン上で簡単に使用できるという事実を紹介するのが目的です。

言語は本来、それぞれに適する分野があり（PL/I とか、C など、アセンブラ・レベルに近いシステム記述から、ビジネス・アプリケーションの記述まで、広く利用できる言語もあるが）、それぞれの特徴を生かしたサンプル・プログラムを作ればなお良いのですが、ここではただ、各言語の書き方や、雰囲気と比較する意味で、画一的に取り扱っていることをご了承下さい。

5.1 COBOL

5.1.1 COBOLについて

COBOL（Common Business Oriented Language）は、その名の通り事務用共通言語であり、身近な例では、銀行のオンライン・システムや、各会社の給与・人事その他の会社経営システムなど、事務分野の大半のものに使用されています。

COBOL は、1959年に事務用共通言語の開発を目的として設立された組織である CODASYL（the Conference On DATA SYstems Language、コダシル）によって、その仕様書が作成され、1960年に

一般に公開されました。

この仕様書に基づき、各社が COBOL コンパイラを開発し、1960年末には、すでに RCA と UNIVAC が完成して両機種間で同一 COBOLプログラムの互換性の実験を行い、見事に成功しています。この当時の COBOL は、現在 COBOL-60と呼ばれています。

CODASYL と ANSI (アメリカ国立標準協会) との関係は、COBOL 文法の開発を CODASYL が行い、その標準化を ANSI が行うことになっています。

COBOL は、CODASYL の管理により常に保守改訂が行われており、他のコンピュータとの互換性が極めて良いことは、他の言語にはない最大の特徴であり、投資効率の高い言語であると言うことができます。

現在の COBOL の規格は、改訂アメリカ規格 COBOL X3.23-R1974が多く使われており、JIS COBOL もこれに準じています。

CODASYL の指示で、COBOL のマニュアルなどには、必ず次の文章が書かれていますので紹介しておきましょう。

COBOL は産業界の言語で、いずれの会社、会社団体、あるいはいかなる組織団体の所有物でもない。

いずれの参画者、CODASYL プログラミング言語委員会も、このプログラミング・システムと言語の正確性および機能に関しては、なんの保証もしないし、一切の責任も負わない。

5.1.2 MICRO FOCUS社 CIS COBOLについて

CIS COBOL は、Federal High COBOL のGSA 仕様に一致しており、CP/M マシン上で（例えばあなたの PC-8001, 8801でも）ANSI X3.23 1974 COBOL の、それもレベルIIを使用することができます（注：ここで使用した CIS COBOL Version 4.4 は、レベルIIのバージョンではありません）。

CIS COBOL は、次の8つのモジュールをレベルIIで完全に実行します。

- ニュークリアス (中核)
- テーブル・ハンドリング (表操作)
- シーケンシャル I/O (順ファイル)
- リラティブ I/O (相対ファイル)
- インデックス I/O (索引ファイル)

- インタープログラム・コミュニケーション（プログラム間連絡）
- ソート・マージ（整列併合）
- コミュニケーション（通信）……ただし、ハードウェアのサポートがされていること（レポート・ライタはない）。

また、次のモジュールはレベル I で実行します。

- セグメンテーション（区分化）
- ライブラリ（登録集）
- デバッグ

その他、ミニコンピュータや、大型コンピュータの COBOL ではサポートされていない COBOL の入出力モジュールに内蔵された、全画面処理用のアドバンス画面形式と、データ入出力機能、カーソルのダイレクト・アドレッシングによるデータ入出力機能、その他、マイクロコンピュータを十分に活用するためのいくつかの拡張機能があります。

また、CIS COBOL には、ユーティリティとして "FORMS-2" という、COBOL ソース・コード・ジェネレータが用意されており、CRT と対話形式で非常に能率的にプログラムを作成することができます。

5.1.3 CIS COBOLによる「SAMPLE」プログラムの作成

まず、そのソース・プログラムを次に示します。

COBOL に関しては、若干 "SAMPLE" プログラムの仕様と異なる箇所もありますが、大形コンピュータで、COBOL の専門家が書くプログラム風に書かれています。もちろん、主題プログラムの仕様を満足するだけであれば、この数分の一のステップ数で書くことができます。

A>TYPE SAMPLE.CBLファイル名のエクステンションは何でもよい。

* CIS COBOL sample program for "Application CP/M" *

IDENTIFICATION DIVISION.

PROGRAM-ID. SAMPLE.

AUTHOR. Mr. MURASE.

DATE-WRITTEN. 8/23/1982

DATE-COMPILED. 8/23/1982

プログラムの識別。

*

ENVIRONMENT DIVISION.

SOURCE-COMPUTER. PC-8801withCPM.

OBJECT-COMPUTER. PC-8801withCPM.

SPECIAL-NAMES. CONSOLE IS CRT.

コンピュータの物理的なものに関することを定義。
→ DISPLAY, ACCEPTで、装置名を省略した時に、
CRTを指定することを定義。

*

DATA DIVISION.オブジェクト・プログラムによって処理されるすべてのデータを、ここで定義する。
 WORKING-STORAGE SECTION.使用する変数はすべて定義する。

77 X PIC 9(6)V9.独立項目としている。01×PIC 9(6)V9. でもよい。
 ↳ 小数点以上6桁、以下1桁を定義。

```

*
01 CRT-TITLE.
03 FILLER PIC X(80) VALUE SPACE.
03 TITLE-A PIC X(80) VALUE "1+2+3+....n = x".
03 FILLER PIC X(80) VALUE SPACE.
03 CRT-INPUT-TITLE.
05 TITLE-B PIC X(30) VALUE
  "input n (n = 1...250) -->000".
05 FILLER PIC X(50) VALUE SPACE.
05 ERR-COMMENT PIC X(80) VALUE SPACE.
03 CRT-ANSWER-AREA.
05 COMMENT-1 PIC X(10).
05 INPUT-NOTE PIC ZZ9.
05 COMMENT-2 PIC X(2).
05 ANSWER PIC ZZ,ZZ9.
05 CUR-SET PIC X(160).
  
```

VALUEで各変数に
初期値を与える。

〜は上位の0を表示しない。
 =は表示する桁で、編集項目である。

.....FILLER以外の変数を定義することで、
 カーソルをエリア最下部にセットする。

```

01 CRT-INPUT-AREA-SET REDEFINES CRT-TITLE.
03 FILLER PIC X(267).
03 INPUT-DATA PIC 9(3).
  
```

*
 PROCEDURE DIVISION.データ処理部。

```

*** INITIAL-SET ***
MOVE ZERO TO X.
DISPLAY SPACE. ....画面全体をクリア。
MOVE SPACE TO CRT-ANSWER-AREA. ....答えエリアをクリア。
  
```

TITLEOUT-ACCEPT.

①→ DISPLAY CRT-TITLE.初期画面表示。
 ②→ DISPLAY CRT-INPUT-AREA-SET.そこにカーソルをセット。
 ③→ ACCEPT CRT-INPUT-AREA-SET.データ入力。

```

*** CHECK DATA ***
IF INPUT-DATA < 251
  THEN MOVE SPACE TO ERR-COMMENT
  MOVE "1+2+3+...." TO COMMENT-1
  MOVE INPUT-DATA TO INPUT-NOTE
  MOVE "=" TO COMMENT-2
  ELSE MOVE "INPUT ERROR ! ----> RETRY PLEASE " TO ERR-COMMENT
  GO TO TITLEOUT-ACCEPT.
  
```

答えエリアの表示内容のセット。
 入力値が正しい時。

入力値エラーの時。

```

*** COMPUTE ***
ADD 1 INPUT-DATA TO X.
MULTIPLY 0.5 BY X.
MULTIPLY X BY INPUT-DATA GIVING ANSWER.
④→ DISPLAY CRT-TITLE. ....画面全体(今回は、答えエリアに内容が入っている)を表示。
  
```

nまでの総和の計算。

STOP RUN.

A>

Figure-5.1.1 COBOL による "SAMPLE" のソース・プログラム。

CIS COBOLのソース・ファイル名のエクステンションは、任意のものでかまいません。

ここでは n までの総和の計算を、ループを使わず、

$$((n+1) \times 0.5) n = x$$

という式で求めています。

スクリーンへの表示は、CRT 全体を各変数で分割して表示位置をセットしています。その変数と CRT の関係を次に示します（このような手法を用いずに、もっと簡単に表示できますが）。

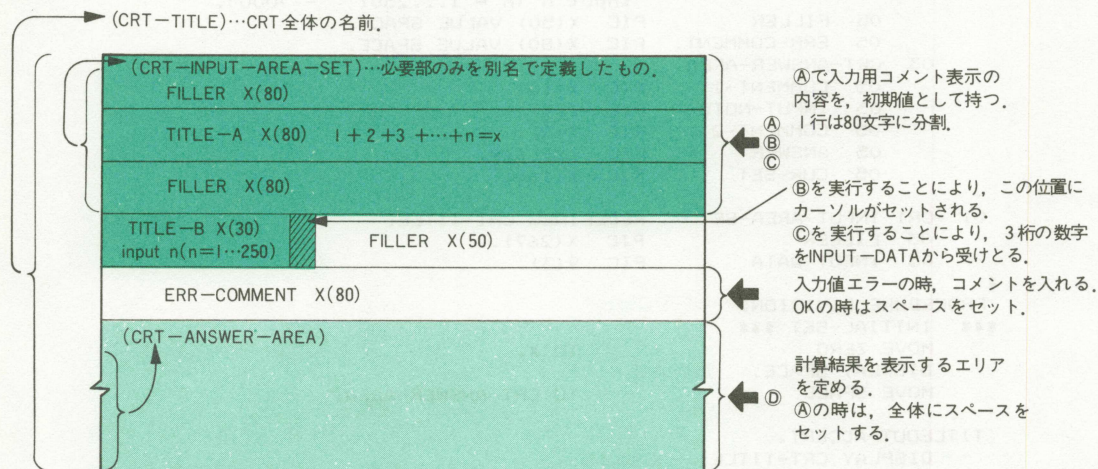


Figure-5.1.2 CRT 全体を各変数で分割した図

さて、次はこのソース・プログラム "SAMPLE.CBL" をコンパイルします。

コンパイルして、そのプログラムを実行するために必要な最低限の各ファイルを、STAT コマンドで示しておきます（もちろん CIS COBOL のパッケージに含まれています）。

A>B:STAT *.*/

Recs	Bytes	Ext	Acc	
272	34k	3	R/W	A:COBOL.COMコンパイラのルート・プログラム.
52	7k	1	R/W	A:COBOL.IO1
91	12k	1	R/W	A:COBOL.IO2
52	7k	1	R/W	A:COBOL.IO3コンパイラのオーパレイ・モジュール.
25	4k	1	R/W	A:COBOL.IO4
272	34k	3	R/W	A:RUN.COMランタイム・システム
20	3k	1	R/W	A:SAMPLE.CBL "SAMPLE"プログラムのソース・ファイル.

Bytes Remaining On A: 140k

A>

Figure-5.1.3 CIS COBOL のコンパイルとプログラムの実行のために、最低限必要な各ファイル。

コンパイラの実行と、生成されたファイルの確認などの実行例を次に示します。

```
A>DIR SAMPLE.* / .....コンパイル前の"SAMPLE"ファイルをすべてリストアウト。

A: SAMPLE   CBL .....ソース・ファイルのみ存在する。

A>COBOL SAMPLE.CBL / .....ソース・ファイル"SAMPLE.CBL"に対し、コンパイラを実行する。

** CIS COBOL V4.4  COPYRIGHT (C) 1978,1981 MICRO FOCUS LTD
**COMPILING SAMPLE.CBL
** ERRORS=00000 DATA=00977 CODE=00303 DICT=00333:09126/09459 GSA FLAGS=  OFF
エラーなしでコンパイル成功。

A>DIR SAMPLE.* / .....コンパイル終了後、すべての"SAMPLE"ファイルをリストアウト。

A: SAMPLE   CBL : SAMPLE   INT : SAMPLE   LST .....INTファイルとLSTファイルが
   ソース・ファイル   中間コード・ファイル   リスト・ファイル   生成されている。

A>B:STAT SAMPLE.INT / .....中間コード・ファイルの容量を調べる。

RECS  BYTES  EXT ACC
  11     2K    1 R/W A:SAMPLE.INT .....2Kバイト長である。
BYTES REMAINING ON A: 131K

A>
```

Figure-5.1.4 CIS COBOL のコンパイラの実行と、生成されたファイルの確認。

コンパイルは成功して、2 Kバイトの中間コードのファイル "SAMPLE.INT" と、リスト・ファイル "SAMPLE.LST" が生成されています。

では、"SAMPLE" プログラムを実行してみましょう。中間コードのプログラムをランタイム・システム "RUN.COM" で実行します。

```
A>RUN SAMPLE.INT / .....コンパイルされた"SAMPLE"プログラムの実行。
                           ランタイム・システムを起動して実行させる。
```

Figure-5.1.5 CIS COBOL で作成された "SAMPLE" プログラムの実行。

ランタイム・システムと、中間コード・ファイルがメモリ上にロードされると実行開始となり、スクリーン全体がクリアされて、次のように表示されます。

```
1+2+3+....n = x                この位置にカーソルが戻る。
input n (n = 1...250)  -->300/ .....251以上を入力したため、エラー・メッセージ
INPUT ERROR !  ----> RETRY PLEASE  が出力されて、元の位置にカーソルが戻った。
```

Figure-5.1.6 "SAMPLE" プログラムの実行①。入力値エラーの例（エラー・メッセージが出力される点は、仕様と異なります）。

前のリストの“300”を“100”と書き換えて実行すると、次のようになりました。

```
1+2+3+....n = x
input n (n = 1...250) -->100! .....適正な値を入力.
1+2+3+....100 = 5,050 .....入力値と答えが出力されている.
A> .....CP Mに戻った.
```

Figure-5.1.7 “SAMPLE”プログラムの実行②。適正な入力値の例。

参考までに、コンパイル時に生成されたリスト・ファイルの“SAMPLE.LST”の一部を次に示しておきます。

```
A>TYPE SAMPLE.LST /
** CIS COBOL V4.4                      SAMPLE.CBL                      PAGE: 0001
**
* CIS COBOL sample program for "Application CP/M" *
IDENTIFICATION DIVISION.
PROGRAM-ID.          SAMPLE.
AUTHOR.              Mr.MURASE.
DATE-WRITTEN.         8/23/1982
DATE-COMPILED.        8/23/1982
*
ENVIRONMENT DIVISION.
SOURCE-COMPUTER.      PC-8801withCPM.
OBJECT-COMPUTER.      PC-8801withCPM.
SPECIAL-NAMES.        CONSOLE IS CRT.
*
DATA DIVISION.
WORKING-STORAGE SECTION.
77 X                  PIC 9(6)V9.
*
01 CRT-TITLE.
   03 FILLER           PIC X(80) VALUE SPACE.
   03 TITLE-A          PIC X(80) VALUE "1+2+3+....n = x".
*
*** COMPUTE ***
ADD 1 INPUT-DATA      TO X.
MULTIPLY 0.5 BY X.
MULTIPLY X BY INPUT-DATA GIVING ANSWER.
DISPLAY CRT-TITLE.
STOP RUN.
** CIS COBOL V4.4 REVISION 1
** COMPILER COPYRIGHT (C) 1978,1981 MICRO FOCUS LTD
** ERRORS=00000 DATA=00977 CODE=00303 DICT=00333:09126/09459 GSA FLAGS= OFF
URN AV/0001/BK
```

Figure-5.1.8 コンパイル時に生成されたリスト・ファイルの一部をタイプアウト。

CIS COBOL のその他多くの機能の紹介は省略します。

5.2 FORTRAN

5.2.1 FORTRANについて

FORTRAN (FORmula TRANslator) は、科学技術計算用の言語であり、1958年に IBM が改訂版の FORTRAN II を発表し、そのコンパイラが作られてから一般に広く普及し始めました。その後、1962年に FORTRAN IV とそのコンパイラが発表され、これが現在の標準化された ANSI FORTRAN の基となっています。

日本の JIS FORTRAN の最高水準7000は、ANSI の FORTRAN に準じています。

FORTRAN は、なぜか日本では非常にもてはやされている言語であり、COBOL に次いで多く使われています。

プログラムの書き方は、BASIC 言語によく似ているので (BASIC が FORTRAN に似ているのであるが)、BASIC 言語の書ける人なら少しの学習で FORTRAN が書けるようになるでしょう。

5.2.2 Microsoft 社 FORTRAN-80について

FORTRAN-80は、ほぼ、JIS FORTRANの水準7000を満たしていますが、いくつかの拡張機能と制約事項があり、それを次に示します。

FORTRAN JIS 水準7000の上に拡張された機能。

- (1) STOP 文、PAUSE 文に "STOP C" または "PAUSE C" と C が使用されるときに、C は 6 文字の ASCII コードが使用できる。
- (2) READ 文もしくは WRITE 文の中に、ERR= および END= と書いて入出力時エラーが発生したとき、および入出力時のファイルが終了したときの分岐を書くことができる。
- (3) PEEK, POKE, INP および OUT 機能が付け加えられている。
- (4) 文関数で添字のついた変数を使うことができる。
- (5) 整数が使用できる場所では、16進数も使用できる。
- (6) 文字データの列 (アポストロフィでくくられた文字) が nH の形の代わりに使用できる。
- (7) 文字データは、整数としての式の中で使用することができる。
- (8) 継続行の数に制約はない。
- (9) 各種の型データを式の中、または代入文に用いることができる。また変換は自動的に行われる。

FORTRAN JIS 水準7000に対して次の制約があります。

- (1) 複素数型のデータが使用できない。
- (2) 宣言文は、つぎの順序になっていなければならない。
 - PROGRAM 文, SUBROUTINE 文, FUNCTION 文, BLOCK DATA 文
 - 型宣言文, EXTERNAL 文, DIMENSION 文
 - COMMON 文
 - EQUIVALENCE 文
 - DATA 文
 - 文関数
- (3) データの型に応じてコンピュータのメモリを占める大きさが異なる。整数, 実数, 倍精度実数および論理型に応じて必要なメモリの容量が異なる。
- (4) 代入文における符号および DO 文の最初のコンマは, その文の開始行に現れなければならない。
- (5) 入出力並びの要素として, 複数のデータをかっこでくくった形は使用できない。

FORTRAN-80は, FORTRAN で書かれたソース・ファイル(ファイル名のエクステンションは"FOR")をコンパイルし, リロケートابل・オブジェクト・ファイル (マイクロソフト・リロケートابل・オブジェクト・フォーマット, エクステンションは"REL")を生成するコンパイラです。生成された REL ファイルは, 付属の LINK-80 (リンク・ローダ) により, FORTRAN のライブラリとのリンク, それにアセンブラや他の言語による REL ファイルとのリンクを行うことができます。

5.2.3 FORTRAN-80による「SAMPLE」プログラムの作成

まず, ソース・プログラムを次に示します。

```
A>TYPE SAMPLE.FOR .....ファイル名のエクステンションは, "FOR"とする.

C      FORTRAN-80 sample program for "Application CP/M" ..... 行の最初が"C"で始まる行は,
C                                                                コメントとなる.
C
C      INTEGER N, NUMB, TEMP .....各変数の定義.
C
C      ** Title out
C      WRITE (1, 100)
100 FORMAT (1H, 15H1+2+3+....n = x/) ..... タイトルの出力.
C
C      ** Input message out, Key input and Check
10 WRITE (1, 200)
200 FORMAT (1H+, 27Hinput n (n = 1...250) --->) ..... 入力メッセージの出力.
C      READ (1, 300) N
300 FORMAT (I10) ..... 数値nの入力とそのチェック.
C      IF (N.LT.1 .OR. N.GT.250) GO TO 10
```



```

C
C      ** Compute
      NUMB = N
      TEMP = N
20    NUMB = NUMB - 1
      IF (NUMB.EQ. 0) GO TO 30
      TEMP = TEMP + NUMB
      GO TO 20
C
C      ** Answer out
30    WRITE (1, 400) N, TEMP
400   FORMAT (1H, 10H1+2+3+...., 16, 3H = , 15//)
      STOP
      END
A>

```

nまでの総和の計算。

答えが表示されている。

Figure-5.2.1 FORTRAN による "SAMPLE" プログラムのソース・ファイル。

ソース・プログラムをコンパイルして、FORTRAN のライブラリとリンクし、実行可能なオブジェクト・ファイルを生成するには、最低限、次に示すファイルが必要です（もちろん FORTRAN-80 のパッケージに含まれています）。

```

A>B:STAT *.* /

```

Recs	Bytes	Ext	Acc
213	27k	2 R/W A:F80.COM	コンパイラ。
207	26k	2 R/W A:FORLIB.REL	リロゲータブル・オブジェクト形式の各種ライブラリ。
84	11k	1 R/W A:L80.COM	リンク・ローダ。
5	1k	1 R/W A:SAMPLE.FOR	"SAMPLE" プログラムのソース・ファイル。

Bytes Remaining On A: 176k

A>

Figure-5.2.2 FORTRAN-80のコンパイル→リンク→実行に必要な最低限の各ファイル。

では、コンパイルの作業を始めましょう。各時点での DIR コマンドによる "SAMPLE" ファイルの確認をしながら、コンパイラの実行とリンク・ローダの実行例を次に示します。

```

A>DIR SAMPLE.* /

```

A: SAMPLE FOR "SAMPLE" ファイルは、ソース・ファイルのみ存在。

```

A>F80 SAMPLE, SAMPLE=SAMPLE /
$MAIN

```

..... コンパイラの実行、標準的なコマンド・ラインであり、REL ファイルと、PRN ファイルをディスク上に生成する。

```

A>DIR SAMPLE.* /

```

..... 生成されたファイルの確認。


```

A: SAMPLE FOR : SAMPLE PRN : SAMPLE REL
   ソース・ファイル      リスト・ファイル      リロータブル・オブジェクト・ファイル
   / 終了と同時にCP/Mに戻る。
A>L80 SAMPLE/E,SAMPLE/N/ .....リンク・ローダの実行。標準的なコマンド・ラインであり、COMファイルが
生成される"COM"ファイルをディスクにセーブする。          ティスク上に生成され、終了後はCP/Mに戻る。
Link-80 3.44 09-Dec-81 Copyright (c) 1981 Microsoft

Data      0103      1B74      < 6769>

24819 Bytes Free
[016F 1B74      27]

A>DIR SAMPLE.* / .....生成されたファイルの確認。

A: SAMPLE FOR : SAMPLE PRN : SAMPLE REL : SAMPLE COM
   実行可能な純マシン・コードのファイルが
   生成されている。

A>B:STAT SAMPLE.COM / ..... "SAMPLE.COM"の
   容量の確認。

RECS  BYTES  EXT  ACC
  53      7K      1 R/W A: SAMPLE.COM .....7K/バイト長である。
BYTES REMAINING ON A: 164K

A>

```

Figure-5.2.3 FORTRAN-80によるコンパイラの実行、リンク・ローダの実行と生成されるファイルの確認。

でき上った "SAMPLE.COM" を実行してみましょう。FORTRAN-80はコンパイラなので、"SAMPLE.COM" 単独で実行できます。

```

A>SAMPLE / ..... "SAMPLE"プログラムの実行。

1+2+3+....n = x
input n (n = 1...250)  -->290 / .....251以上のため入力エラーとなる。
input n (n = 1...250)  -->200 / .....再度正しい値を入力。

1+2+3+.... 200 = 20100 .....入力した値と答えが出力された。

STOP .....FORTRAN-80のシステムからの出力。

A>

```

Figure-5.2.4 FORTRAN-80で作成された "SAMPLE" プログラムの実行。

参考までに、リスト・ファイルの "SAMPLE.PRN" の一部を示しておきます。

```

A>TYPE SAMPLE.PRN /

FORTRAN-80 Ver. 3.44 Copyright 1978-1981 (C) By Microsoft -- Bytes: 16966
Created: 10-Dec-81
1      C      FORTRAN-80 sample program for "Application CP/M"

```


5.3 BASIC

5.3.1 BASICについて

BASIC は、もう言わずと知れた BASIC であり、パソコン=BASIC、世の中のコンピュータはすべて BASIC、と思い込んでしまっている人もいるくらいであり、それほど普及している言語です。

BASIC の名は、一説によると Beginner's All-purpose Symbolic Instruction Code に由来するとされていますが、筆者個人的にはただ、「基礎の」とか「初歩の」といった意味での "basic" をとったものである方が "BASIC" らしいのではないかと考えています。

BASIC は、米国 Dartmouth 大学の J. G. Kemeny と、T. E. Kurtz を中心に開発された、会話型のプログラミング言語であり、FORTRAN に非常によく似ています。一般に使われ始めたのは、1965 年に GE235 システムにインプリメントされてからであると言われています。

開発当初は、大型マシンの TSS で使用する目的であったのですが、マイクロコンピュータの驚異的な発達のおかげで、今では各家庭やオフィスのパーソナル・コンピュータで主に使用されており、言語の機能も、当初のスーパー・スーパー・セットと言えるほど拡張されています。

BASIC のコンパイラは、マイクロコンピュータ用のものが実現されており、BASIC も実務レベルで本格的に利用できるようになりました。コンパイラ型 BASIC については、本書 3.4 章でマイクロソフト社の "BASCOM" を、当項でデジタルリサーチ社の "CB-80" を取り上げています。

5.3.2 Digital Research 社 CB-80 について

(CB-80 は Compiler System 社の製品であるが、1981 年秋に同社は Digital Research 社と合併した)

CB-80 は、アメリカでビジネス・ユースとして最も広く使用されている CBASIC (Digital Research 社、事務用の中間コード・インタープリタ型 BASIC) とソース・ファイルのコンパチビリティを持った、BASIC コンパイラです。

CB-80 は、CB-80 コンパイラとライブラリ、それにリンク・ローダから構成され、BASIC 言語のソース・プログラムを、マシン・コードのリロケャブル・オブジェクト・モジュールにコンパイルします。生成されたリロケャブル・オブジェクト・モジュールは、リンク・ローダにより、CB-80 のライブラリ・モジュールや、必要であれば他の言語から生成されたモジュールとの結合を行い、実行可能な 1 本のマシン・コードプログラムを生成します。

CB-80 は、従来の BASIC にはない、数々の特徴を持っています。

- ラインNoを一切必要としない。GOTO および GOSUB にはラベルを使用できる（ラインNoを付けてもかまわないが）。
- 従来、1行に記述しなければならなかったものが、複数行にわたって記述できる。よって、複数行にわたる関数も自由に定義できる。
- リロケータブル・マクロ・アセンブラで使用するものと同じ機能を持つ“PUBLIC”（3.2、3.3章参照）宣言を、他のモジュールに対して行える。
- ファイルのランダム・アクセスの手続きが簡単である。数値を文字列に変換する必要がある。
- %INCLUDE 文により、別のソース・ファイルを任意の場所に挿入しながらコンパイルを行うことができる。
- その他いくつかありますが省略。

などであり、何と言っても最大の利点は、アセンブラ言語を記述する感覚で、BASIC 言語を記述できるということでしょう。そのため、構造化プログラミングが格段に行いやすくなっています。

従来のラインNo付きの BASIC では、とても書けなかったり、書く気にはなれなかったプログラムでも、CB-80の記述法でなら可能となるものが多いと思われます。

注) 巻末の付録Bに、CBASIC, CB-80, MBASIC, それに BASCOM のステートメントおよび関数の一覧表を載せてあります。参照下さい。

5.3.3 CB-80による「SAMPLE」プログラムの作成

まず、そのソース・プログラムを示します。ファイル名のエクステンションは、“BAS” がデフォルトとして指定されていますが、何であっても構いません。

ラインNo. が全くないと、自由な書き方に注目。
 A>TYPE SAMPLE.BASファイル名のエクステンションは何でもよいが、“BAS”がデフォルト名である。
 * CB-80 sample program for "Application CP/M" *行の“*”あるいは“\”のあとは無視される。
 INTEGER I,H,AFORTRAN風の整数宣言。
 *input message outこのように自由にコメントを書く。
 PRINT
 PRINT "1+2+3+...n = x"
 INP.MSGOUT:ラベルを使用できる。ラベルにピリオドを含めてもよい。
 INPUT "input n (n = 1...250) -->" ; INPVAL
 IF (INPVAL<1) OR (INPVAL>250)
 THEN GOTO INP.MSGOUT
 *compute
 GOSUB CALC.SUBラベルでサブルーチンを呼んでいる。
 *out result
 PRINT

“*”あるいは“\”により、同一行に書かなければならないものを、このように複数行に分けて書くことができる。


```

PRINT "1+2+3+...." ; INVAL ; "= " ; ANSX
STOP      *program run end .....プログラムの実行の終わりはENDではなくSTOP.

      *calc subroutine
CALC.SUB:.....ラベル
      P=      INVAL+1
      ANSX=    P * 0.5 * INVAL
      RETURN

*List end.
A>

```

Figure-5.3.1 BASIC コンパイラ CB-80で書いた "SAMPLE" プログラムのソース・ファイル.

このソース・プログラムをコンパイルし、ランタイム・ライブラリをリンクして実行するには、CB-80のパッケージの中から、最低限次の各ファイルが必要です。それらを STAT コマンドで次に示します。

```

A>B:STAT *.* /

Recs  Bytes  Ext  Acc
  41    6k    1 R/W A:CB80.COM .....コンパイラのルート・プログラム.
 106   14k    1 R/W A:CB80.OV1 .....コンパイラのオーバーレイ.
 106   14k    1 R/W A:CB80.OV2 .....コンパイラのオーバーレイ.
 125   16k    1 R/W A:CB80.OV3 .....ランタイム・ライブラリ
 156   20k    2 R/W A:CB80.IRL .....リンク・ローダ.
   55    7k    1 R/W A:LK80.COM .....リンク・ローダ.
   4    1k    1 R/W A:SAMPLE.BAS ..... "SAMPLE"のソース・プログラム.

Bytes Remaining On A: 163k

A>

```

Figure-5.3.2 CB-80のコンパイル→リンク→実行に必要な最低限の各ファイル.

では、コンパイラを実行します。ソース・ファイル名のエクステンションが "BAS" の場合は、 "BAS" を省略できますが、 "BAS" 以外の場合は、フルネームでソース・ファイル名を指定します。

```

A>DIR SAMPLE.* / ..... "SAMPLE"に関するファイルを調べる.

A: SAMPLE  BAS ..... ソース・ファイルのみ存在している.

A>CB80 SAMPLE[S] / ..... ソース・ファイル "SAMPLE.BAS" をコンパイル.
                        [S]により、RELファイル中にシンボル・ロケーション情報を含める.

-----
CB80 Version 1.3 Serial No. 072-0000 Copyright (c)
1981 Digital Research, Inc. All rights reserved
-----

```


コンパイラ実行中の各パスが表示され、リスト・ファイルがコンソールに出力される。

```

end of pass 1
end of pass 2
1: 005ch      * CB-80 sample program for "Application CP/M" *
2: 005ch
3: 005ch
4: 005ch      INTEGER I,H,A
5: 005ch
6: 005ch      %input message out
7: 005ch      PRINT
8: 005fh      PRINT "1+2+3+....n = x"
9: 0065h      INP.MSGOUT:
10: 0065h      INPUT "input n (n = 1...250) -->"; INPVAL
11: 0076h      IF (INPVAL<1) OR (INPVAL>250)
12: 0097h      THEN GOTO INP.MSGOUT
13: 009ah
14: 009ah      %compute
15: 009ah      GOSUB CALC.SUB
16: 009dh
17: 009dh      %out result
18: 009dh      PRINT
19: 00a0h      PRINT "1+2+3+...."; INPVAL; "= "; ANSX
20: 00beh
21: 00beh      STOP %program run end
22: 00beh
23: 00c1h
24: 00c1h      %calc subroutine
25: 00c1h      CALC.SUB:
26: 00c1h      P= INPVAL+1
27: 00d3h      ANSX= P * 0.5 * INPVAL
28: 00e8h      RETURN
29: 00ech
30: 00ech %List end.
end of compilation
no errors detected
code area size: 236 00ech
data area size: 24 0018h
common area size: 0 0000h
symbol table space remaining: 16423

A>コンパイル成功。エラーなし。

```

Figure-5.3.3 CB-80によるコンパイラの実行。

次はコンパイルにより生成されたリロケータブル・オブジェクト・ファイル "SAMPLE, REL" と、ランタイム・ライブラリ "CB-80, IRL" とのリンクを行います。必要ならば、他の言語や、アセンブラからのリロケータブル・オブジェクトとのリンクも自由に行うことができます。

A>DIR SAMPLE.* /コンパイルにより生成されたファイルの確認。

A: SAMPLE BAS : SAMPLE REL
ソース・ファイル リロケータブル・オブジェクト・ファイル。

A>LK80 SAMPLE /リンク・ローダの実行。"SAMPLE, REL"と"CB80, IRL"をリンクする。


```
-----
LK80 VERSION 1.3 SERIAL NO. 07245678 COPYRIGHT (C)
1982 DIGITAL RESEARCH, INC. ALL RIGHTS RESERVED
-----
```

```
CODE SIZE:      1680 (0100-177F)
COMMON SIZE:    0000
DATA SIZE:      01AC (1780-192B)
SYMBOL TABLE SPACE REMAINING:  75D9
```

A>DIR SAMPLE.* / -----生成されたファイルの確認.

```
A: SAMPLE  BAS : SAMPLE  REL : SAMPLE  SYM : SAMPLE  COM
                                     シンボル・ロケーション・ファイル 実行可能な純マシン・コード・ファイル.
```

A>B:STAT SAMPLE.COM / -----"SAMPLE.COM"の容量を調べる.

```
RECS  BYTES  EXT ACC
  45    6K    1 R/W A: SAMPLE.COM -----6K/バイト長である.
BYTES REMAINING ON A: 152K
```

A>

Figure-5.3.4 "SAMPLE" ファイルの確認と、ランタイム・ライブラリとのリンク.

以上の手順で、実行可能なオブジェクト・ファイル "SAMPLE.COM" ができ上がりました。
では、このプログラムを実行してみましょう。

A>SAMPLE / -----"SAMPLE"プログラムの実行.

```
1+2+3+....n = x
input n (n = 1...250)  --> 500 / -----251以上で入力エラー.
input n (n = 1...250)  --> 250 / -----今回は正しい入力値.

1+2+3+.... 250 = 31375 -----入力値と答えが表示されている.
```

A>

Figure-5.3.5 BASIC コンパイラ CB-80で作成された "SAMPLE" プログラムの実行.

参考までに、リンク・ロードの実行で生成されたシンボル・ロケーション・ファイルをタイプアウトして示します。この "SYM" ファイルは、デバッガの SID や ZSID (3.1 章参照) のシンボル入力用ファイルとして使用できます。

A>TYPE SAMPLE.SYM /シンボル・ロケーション・ファイルのタイプアウト.

16EB INP.MS 1914 INPVAL 1744 CALC.S 191C ANSX
1924 P

A>

Figure-5.3.6 リンク・ローダの実行により生成されたシンボル・ロケーション・ファイルをタイプアウト.

参考までに、PC-8801上のN-BASICとPC-8801上のCP/MでのCB-80とを、簡単なプログラムを使って処理速度の比較を行ってみました.

プログラムは、変数Pを1から100まで変化させ、その2乗を求めて変数Xとし、それを何回か繰り返し、PとXを刻々とスクリーンに表示させる、という簡単なものです.

全く同一内容のこのプログラムをN-BASICとCB-80とで、実行したものを次に示します.

load "bench" /N-BASICでの"bench"プログラムをロード.

Ok

list /プログラムをリストアウト.

```
10 DEFINT P,L,Y,X
20 PRINT TIME$:PRINT
30 FOR P=1 TO 100
40 FOR L=1 TO 100:Y=P/2:X=(Y+Y)*P:NEXT L
50 PRINT P;X;CHR$(&HD);
60 NEXT P
70 PRINT:PRINT TIME$
80 END
```

Ok

run /実行.(PC-8801のN-BASICによる)

00:00:22プログラムのスタート時刻.

100 10000次々と変化するPとXの値. 所要時間2分05秒

00:02:27プログラムの終了時刻.

Ok

CB-80と全く同一のプログラム.
ただしインタープリタでの実行スピード向上のために、マルチステートメントなどの配慮がされている.

Figure-5.3.7 PC-8801のN-BASICでの実行.

A>TYPE BENCH.BAS /CB-80での"BENCH"プログラムのソース・ファイルをタイプアウト.

INTEGER P,L,Y,X

PRINT CHR\$(1BH); "A"NECのCP/Mでの時刻表示のエスケープ・シーケンス.

PRINT

P=0 : L=0 : X=0


```
FOR      P=1 TO 100
FOR      L=1 TO 100
      Y=P/2
      X=(Y+Y)*P      *SAME AS ( )
NEXT      L
PRINT    P ; X ; CHR$(ODH) ;
NEXT      P          復帰のみ行う
PRINT
PRINT
PRINT    CHR$(1BH) ; "A" .....前出.

STOP
```

A>

Figure-5.3.8 CB-80のソース・ファイル.

A>BENCH!CB-80でコンパイルした"BENCH"プログラムの実行.

00:11:32プログラムのスタート時刻.

100 10000次々と変化するPとXの値.

所要時間6秒
(N-BASICの約21倍)

00:11:38プログラムの終了時刻.

00:11:38CP/Mのレポートによる時刻表示.

A>

Figure-5.3.9 PC-8801+NEC CP/M での CB-80の実行.

CB-80の、その他の多くの機能の紹介は省略します.

5.4 PASCAL

5.4.1 PASCALについて

PASCAL は、1968～1970年に、チューリッヒの Niklaus Wirth によってスイスで作られました。PASCAL という名は、フランスの数学者、Blaise Pascal (1623～1662) の名をとったもので、他の多くの言語のように、語の頭文字を綴ったものではありません。

最初の PASCAL コンパイラは、1970年に Control Data 社の大型マシン6600システムにおいて実用化されています。

PASCAL は、本章の5.10に取り上げた ALGOL 60 がその母体であると言われ、文法などよく似ています（互いのソース・プログラムを比較参照してみてください）。PL/I や COBOL のような煩雑

さがなく、BASIC や FORTRAN の欠点であるストラクチャード・プログラミングという点での不足を補った、最も学習しやすく使いやすい言語の1つです。

5.4.2 Digital Research社 Pascal/MT+について

(Pascal/MT+は、MT Microsystem 社の製品であるが、同社は1981年秋に、Digital Research 社と合併した)

Pascal/MT+は、コンパイラやエディタなどの言語システム、それにビジネス・パッケージなどのデータ処理用のアプリケーションを書いたり、一方、計測システムや通信関係のリアル・タイム処理用アプリケーションを書いたり、それら双方の目的に使用できるコンパイラです。

Pascal/MT+は、ISO 標準 (DPS/7185) のスーパーセット (ISO 標準を満足して、さらに拡張機能がある) であり、ROM 化も可能な純マシン・コードを出力します。よって、ユーザーにとってはあまり意味のないPコードを出力する PASCAL コンパラなどより、実行速度が5～10倍高速になります。

ISO 標準の拡張としては、モジュラー・コンパイル、255文字までの文字列操作、アセンブリ言語とのリンク、アドレスおよびサイズの関数、割り込み処理などがあります。

また、I/O ポートのコントロールが可能であり、かつ CP/M の BDOS を利用しないマシン・コードを生成することができ、ROM 化が可能なので、制御用プログラムとして機器に組み込むことができます。

また、開発ツールとして、コンパイラ、リンカ、シンボリック・デバッガ、逆アセンブラが含まれており、開発環境が非常に良い PASCAL システムであると言えます。

5.4.3 Pascal/MT+による「SAMPLE」プログラムの作成

まず、そのソース・プログラムを次に示します。

```
A>TYPE SAMPLE.SRC .....ファイル名のエクステンションは何でもよいが、"SRC"がデフォルト名である。

      (* Pascal/MT+ sample program for "Application CP/M" *)
      (* ~~~~~あるいは|~|の間がコメントとなる。~~~~~ *)

program sample;                                (* program name *)

var n, numb, temp: integer;                    (* variable declaration *)

begin
  writeln;                                     (* print title *)
  writeln ('1+2+3+....n = x');
  repeat
    write ('input n (n = 1...250)  -->'); (* print prompt message *)
    read (n);                                (* input n *)
  until (n > 0) and (n <= 250);              (* range check *)
  writeln;
```

内容の説明はこれらのコメントを参照。


```

numb := n; temp := n;                                (* compute summation *)
while numb > 1 do
begin
    numb := numb - 1;
    temp := temp + numb
end;

writeln ('1+2+3+....', n, ' = ', temp)                (* print result *)
end. .... " を忘れないように.
```

このプログラムは小文字で書いてあるが、大文字で書いても構わない(しかし小文字の方が美しい).

A>

Figure-5.4.1 Pascal/MT+で書いた「SAMPLE」プログラムのソース・ファイル.

ソース・ファイル名のエクステンションは何でも構いませんが, "SRC" がデフォルト名となっているので, "SRC" としておけば, コンパイルやリンクなどの実行時のコマンド・ラインで省略ができて便利です.

次に, Pascal/MT+のパッケージの中から, "SAMPLE" プログラムをコンパイル→リンク→実行するのに必要な最低限の各ファイルを, STAT コマンドで示しておきます.

A>B:STAT *.* /

Recs	Bytes	Ext	Acc	
278	35k	3	R/W	A:MTPLUS.COMコンパイラのルート・プログラム.
100	13k	1	R/W	A:MTPLUS.000
84	11k	1	R/W	A:MTPLUS.001
55	7k	1	R/W	A:MTPLUS.002
59	8k	1	R/W	A:MTPLUS.003
136	17k	2	R/W	A:MTPLUS.004
61	8k	1	R/W	A:MTPLUS.005
46	6k	1	R/W	A:MTPLUS.006デバツガ用のルーチンの1つ.ここでは特に必要ではない.
90	12k	1	R/W	A:LINKMT.COMリンク・ローダ.
190	24k	2	R/W	A:PASLIB.ERLランタイム・ライブラリ.
7	1k	1	R/W	A:SAMPLE.SRC「SAMPLE」のソース・ファイル.
145	19k	2	R/W	A:DIS8080.COM逆アセンブラ.ここでは特に必要ではない.

Bytes Remaining On A: 80k

A>

Figure-5.4.2 Pascal/MT+のコンパイル, リンクに必要な最低限の各ファイル.

では, ソース・ファイル "SAMPLE.SRC" のコンパイルから始めます. コンパイラの実行と, 生成されたファイルの確認の実行例を次に示します.

A>DIR SAMPLE.* /“SAMPLE”ファイルの確認。

A: SAMPLE SRC最初はソース・ファイルのみ存在。

A>MTPLUS SAMPLE \$XPA /コンパイラの実行。スイッチ“\$XPA”は、逆アセンブラのための
オブジェクト・コードと、リスト・ファイルの生成することを指示している。

Pascal/MT+ Release 5.5
(c) 1981 MT MicroSYSTEMS, Inc.

Disassembler records enabled
PRN file routed to disk: A
CP/M-80 version
+
Source lines: 25
Symbol Table Initialization
Available Memory: 6109
User Table Space: 2113
V5.5 Phase 1

Remaining Memory: 2085
V5.5 Phase 2
SAMPLE

Lines : 25
Errors: 0
Code : 283
Data : 6
Pascal/MT+ 5.5 Compilation Complete

コンパイル成功。エラーなし。

A>DIR SAMPLE.* /生成されたファイルの確認。

A: SAMPLE SRC : SAMPLE PRN : SAMPLE ERL
ソース・ファイル リスト・ファイル リロケートブル・オブジェクト・ファイル
A>

Figure-5.4.3 Pascal /MT+のコンパイルの実行と生成されたファイルの確認。

コンパイルは成功して、“SAMPLE.ERL”（ERL=Extended ReLocatable object code）が生成されています。“ERL”ファイルは、マイクロソフト社の“REL”ファイルの上位コンパチブルのリロケートブル・オブジェクト・ファイルで、Pascal/MT+で使用する逆アセンブラやデバッガのための情報まで含んでいます。

次の作業は、“SAMPLE.ERL”と“PASLIB.ERL”をリンクして、実行可能なオブジェクト・ファイルの生成するために、リンク・ローダを実行します。その実行例を次に示します。

A>LINKMT SAMPLE,PASLIB/S/W /リンク・ローダの実行。スイッチ“/S”は、必要なライブラリのみを選択する指示、“/W”はSIOやZSIOのための“SYM”ファイルの生成する指示。

Link/MT+ Release 5.5

Processing file- SAMPLE .ERL


```

Processing file- PASLIB .ERL

Undefined Symbols:

No Undefined Symbols

0051 (decimal) records written to .COM file

Total Data: 03C9H bytes
Total Code: 193AH bytes
Remaining : 5DB8H bytes

Link/MT+ Release 5.5 processing completed
リンク成功.
A>DIR SAMPLE.* / .....生成されたファイルの確認.

A: SAMPLE SRC : SAMPLE PRN : SAMPLE ERL : SAMPLE SYM
A: SAMPLE COM SID:ZSID用のシンボル・テーブル・ファイル.
実行可能な純マシン・コードファイル.
A>B:STAT SAMPLE.COM / .....“SAMPLE.COM”のファイル容量を調べる.

Recs Bytes Ext Acc
51 7k 1 R/W A:SAMPLE.COM .....7K/バイト長である.
Bytes Remaining On A: 67k

A>

```

Figure-5.4.4 リンク・ローダの実行、実行可能なオブジェクト・ファイルを生成する。

リンクも成功して、純マシン・コードの“COM”ファイルが生成されました。同時に、スイッチ“W”により、SID, ZSID（シンボリック・インストラクション・デバッグ、3.1章参照）に入力するシンボル・テーブル・ファイルも生成されています。

```

A>TYPE SAMPLE.SYM / .....シンボル・テーブル・ファイルをタイプアウトする.

199E MOVERI      1981 MOVELE      19C0 FILLCH      1239 IORESU
1981 MOVE        1247 RESULTI    0C2B GET         0D54 PUT
1A3B SYSMEM      02BF INPUT      0382 OUTPUT      021F N
021D NUMB        021B TEMP

A>

```

Figure-5.4.5 生成された、SID, ZSID 用の“SYM”ファイルのタイプアウト。

では、でき上ったサンプル・プログラムを実行してみましょう。

A>SAMPLE / "SAMPLE"プログラムの実行.

```
1+2+3+....n = x
input n (n = 1...250)  -->400/ ..... 251以上の入力値エラー.
input n (n = 1...250)  -->150/ ..... 今回は正しい値.

1+2+3+....150 = 11325 ..... 入力値と答えが出力されている.
```

A>

Figure-5.4.6 Pascal/MT+で作成された "SAMPLE" プログラムの実行.

参考までに、Pascal/MT+の多くの機能の内の1つである "逆アセンブラ" を実行してみましょう。このユーティリティ・プログラムは、コンパイルにより生成された "ERL" と "PRN" ファイルから、アセンブリ言語のソース・プログラムを作成します。

入力ファイル名.
 ↓
 出力ファイル名.

```
A>DISBOBO SAMPLE DISASM.LST /

Pascal/MT+  --Disassembler--  5.5

CP/M-80 version

Object code file: SAMPLE.ERL
Listing file      : SAMPLE.PRN
Output file       : DISASM.LST

.....
End of program

A> この時点で、"DISASM.LST"が作成されている.
```

Figure-5.4.7 pascal/MT+のユーティリティ・プログラムの1つ "逆アセンブラ" の実行.

作成されたアセンブリ言語のソース・プログラム "DISASM.LST" の一部を次にタイプアウトして示します。

```
A>TYPE DISASM.LST /

Pascal/MT+  Release 5.5  Copyright (c) 1981 by MT MicroSYSTEMS  Page #   1
Disassembly of: SAMPLE

Stmt  Nest      Source Statement / Symbolic Object Code

      TEMP      EQU      0000
      NUMB      EQU      0002
      N         EQU      0004
1      0        (* Pascal/MT+  sample program for "Application CP/M"  *)
2      0
```



```

3      0
4      0      program sample;                                (* program name *)

0000      DB      00,00,00,00,00,00,00,00
0008      DB      00,00,00,00,00,00,00,00
0010      JMP      0000

5      0
6      0      var  n, numb, temp: integer;                  (* variable declaration *)
7      1
8      1      begin

0013      LHLD      0006
0016      SPHL
0017      CALL      0000
          .
          .
          .
0112      CALL      00FC'

25      1      end.

0115      CALL      00A7'
0118      CALL      0000

External reference chain @WIN      --> 0113
External reference chain @RIN      --> 007F
External reference chain @CRL      --> 0116
External reference chain @GTI      --> 00BE
External reference chain @LEI      --> 0095
External reference chain @SFB      --> 00DE
External reference chain @DWD      --> 0110
External reference chain @INI      --> 0018
External reference chain @WRS      --> 0109
External reference chain @HLT      --> 0119
External reference chain OUTPUT    --> 00DA
External reference chain INPUT      --> 0078

A>

```

Figure-5.4.8 “逆アセンブラ”で作成された、アセンブリ・ソース・プログラムのタイプアウトの一部。

Pascal/MT+の、その他の多くの機能の紹介は省略します。

5.5 PL/I

5.5.1 PL/Iについて

PL/I (Programming Language One) は、1965年に IBM から、まずその仕様書 System 360 Operating System PL/I Language Specification が一般に公表されました。

1966年には、その仕様に基づき、最初のコンパイラがイギリスの IBM Hursley 研究所で作られ、以後、何度か機能の拡張が行われ今日に至っています。

PL/Iは、それが計画された当初の目的通り、COBOL の持つ事務処理能力と FORTRAN や ALGOL

の持つ数学的な処理能力を合わせ持ち、かつ、アセンブラ・レベルの処理をも可能にした、非常に広い分野に適合できるプログラミング言語です。PL/Iは、構造化プログラミング言語であり、そのプログラムは、読み易くて書き易いという特徴があり、今日のプログラミング言語の中でも重要な位置を占めています。

5.5.2 Digital Research社 PL/I-80について

PL/I-80は、CP/Mの開発者でデジタルリサーチ社の社長である Dr. Gary Kildall 自らが、情熱を持って作りあげたコンパイラです。

PL/Iは非常に大きなシステムであり、ミニコンピュータにとっても大き過ぎ、そのフルセットは実現が困難です。よって、ANSIは、PL/IサブセットGを設定しており、DEC、Data General、Wangなどのミニコンピュータでは、この仕様に準拠したPL/Iを提供しています。当PL/I-80も同じくサブセットGに準拠しており、これは他のサブセットGおよびフルセットのPL/Iに対して、アップ・コンパチブルです。

PL/I-80の特徴を列記すると、

- 事務計算のための15桁10進計算。高速な科学技術計算のための固定および浮動2進データ形式、それに配列とポインタ変数。
- 文字列とビット列操作。
- シーケンシャルおよびランダム・ファイルをサポート。
- LINKによるオーバーレイ。

などがサポートされており、コンパイルにより、マイクロソフト・フォーマットのリロケータブル・オブジェクト・モジュールを生成します。よって、他の言語で作られたライブラリ・モジュールとの結合が可能です。

4.2章で取り上げた8080→8086トランスレータ“XLT86”は、このPL/I-80によって書かれています。

5.5.3 PL/I-80による「SAMPLE」プログラムの作成

まず、PL/I-80による“SAMPLE”プログラムのソース・ファイルを示します。ファイル名のエクステンションは、必ず“PLI”でなければなりません。

A>TYPE SAMPLE.PLIソース・ファイル名のエクステンションは“PLI”でなければならない。

```
/* PL/I-80 sample program for "Application CP/M" */
/* ~~~~~ の間がコメントとなる。
sample: .....このプログラムを"sample"とする。
```



```

proc options(main); .....プログラムの開始.
dcl
    (n,numb) fixed decimal(4), .....4桁の10進固定小数点.
    temp fixed decimal(6); .....6桁の10進固定小数点.
/* title out */
    put skip list('1+2+3+....n = x'); .....復帰・改行とタイトルの出力.
    put skip; .....復帰・改行.

/* input message out and key input */
keyinp:
    put list('input n (n = 1...250)  -->'); .....入力メッセージ出力.
    get list (n); .....数値の入力.

    if (n<1) ! (n>250)
        then go to keyinp; .....入力値の判定.
    else

/* compute */
    temp = 0;
    do numb = 1 to n; .....nまでの総和の計算.
        temp = temp + numb;
    end;

/* answer out */
    put skip list('1+2+3+....',n,'=',temp); .....答えの出力.

end sample;
A>

```

Figure-5.5.1 PL/I-80 による "SAMPLE" プログラムのソース・ファイル。

次に、PL/I-80 のパッケージの中から、"SAMPLE"プログラムを開発するのに最低限必要な各ファイルを示しておきます。

```

A>B:STAT *.*/

```

Recs	Bytes	Ext	Acc	
60	8k	1	R/W	A:PLI.COMPL/Iコンパイラのルート・プログラム.
142	18k	2	R/W	A:PLIO.OVLコンパイラのオーバーレイ.
254	32k	2	R/W	A:PLI1.OVL
247	31k	2	R/W	A:PLI2.OVL
122	16k	1	R/W	A:LINK.COMリンク・ローダ.
347	44k	3	R/W	A:PLILIB.IRLランタイム・ライブラリ.
5	1k	1	R/W	A:SAMPLE.PLI "SAMPLE"プログラムのソース・ファイル.

```

Bytes Remaining On A: 91k
A>

```

Figure-5.5.2 PL/I-80 による "SAMPLE" プログラムの開発に、最低限必要な各ファイル。

では、コンパイルの作業から始めます。コンパイラの実行と生成されたファイルの確認の実行例を次に示します。


```

A>DIR SAMPLE.*! .....最初の“SAMPLE”ファイルの確認。

A: SAMPLE   PLI .....ソース・ファイルのみ存在。

A>PLI SAMPLE $DI! .....コンパイラの実行。アセンブリ・ソース・コードに展開した“PRN”ファイルを
                        ディスクにセーブするためのスイッチ($DI)を付けた。
      NO ERROR(S) IN PASS 1

      NO ERROR(S) IN PASS 2
コンパイル成功。エラーなし。

A>DIR SAMPLE.*! .....生成されたファイルの確認。

A: SAMPLE   PLI : SAMPLE   PRN : SAMPLE   REL
   ソース・ファイル      生成された      生成された
   リスト・ファイル      リロケートブル・オブジェクト・ファイル
A>

```

Figure-5.5.3 PL/I-80 によるコンパイラの実行と生成されたファイルの確認。

コンパイルは成功して、“REL”リロケートブル・オブジェクト・コードと、コンパイル時のコマンド・ラインに付けた“\$DI”スイッチにより、アセンブリ言語に展開された“PRN”ファイルが生成されました。

次は、リンク・ローダで、“SAMPLE.REL”に、各種ランタイム・ライブラリ (PLILIB, IRL) をリンクして、実行可能な1本のファイルにする作業です。ここで使用するリンク・ローダは、PL/I-80のパッケージに含まれている、デジタルリサーチ社のリンク・ローダです。その実行例を次に示します。

```

A>LINK SAMPLE ! .....リンク・ローダの実行。

LINK 1.3

PLILIB  RQST  SAMPLE  0100  /SYSIN/  1EE3  /SYSPRI/  1F08 .....シンボル・テーブル
                                                が出力される。
ABSOLUTE      0000
CODE SIZE     1DBA (0100-1EB9)
DATA SIZE     0222 (1F5E-217F)
COMMON SIZE   00D4 (1E8A-1F5D)
USE FACTOR    E0
リンク成功。

A>DIR SAMPLE.*! .....生成されたファイルの確認。

A: SAMPLE   PLI : SAMPLE   PRN : SAMPLE   REL : SAMPLE   COM
   ソース・ファイル      生成された      生成された      生成された実行可能な
   リスト・ファイル      リロケートブル・オブジェクト・コード・ファイル。
A>B:STAT SAMPLE.COM ! .....“SAMPLE.COM”のファイルの容量を調べる。

```



```

Recs  Bytes  Ext Acc
  65    9k    1 R/W A: SAMPLE.COM -----9k/バイト長である.
Bytes Remaining On A: 76k

A>

```

Figure-5.5.4 コンパイルにより生成された "SAMPLE. REL" とランタイム・ライブラリとのリンク.

最終目的である実行可能な1本の純マシン・コードのオブジェクト・ファイルができ上がりました。同時に、SID や ZSID (3.1章参照) に入力可能なシンボル・テーブル・ファイルも作られています。では、でき上がった "SAMPLE" プログラムを実行してみましょう。

```

A>SAMPLE / -----"SAMPLE"プログラムの実行.

1+2+3+....n = x
input n (n = 1...250)  -->300 / -----入力値のエラー.
input n (n = 1...250)  -->200 / -----今回は正しい値を入力.

1+2+3+.... 200 = 20100 -----入力値と答えが出力された.
End of Execution -----PL/Iのシステムによる出力.
A>

```

Figure-5.5.5 PL/I-80 で作成された "SAMPLE" プログラムの実行.

次に参考までに、コンパイル時に生成されたりスト・ファイル "SAMPLE. PRN" の一部を示しておきます。PL/Iのソース・プログラムが、アセンブリ言語に展開されています。

```

A>TYPE SAMPLE.PRN /

PL/I-80 V1.3  COMPILATION OF: SAMPLE

D: Disk Print
I: Interlist Source and Code

NO ERROR(S) IN PASS 1

NO ERROR(S) IN PASS 2

PL/I-80 V1.3  COMPILATION OF: SAMPLE

1  0000 /* PL/I-80 sample program for "Application CP/M" */
2  0000
3  a 0000 sample:
      0000     LXI  B,0200
      0003     CALL ?START
4  a 0006     proc options(main);
5  c 0006     dcl

```



```

6 c 0006      (n,numb) fixed decimal(4),
7 c 0006      temp fixed decimal(6);
8 c 0006
9 c 0006      /* title out */
10 c 0006      put skip list('1+2+3+....n = x');
    0006      LXI D,0252
    0007      LXI B,0000
    000C      CALL ?SYSPR
        .
        .
        .
    0154      MVI A,09
    0156      MVI B,00
    0158      CALL ?QDCOP
    015B      CALL ?PNVOP
    015E      CALL ?QIOOP
    011A      ==== 0161
29 c 0161
30 a 0161 end sample;
    0161      CALL ?STOPX

CODE SIZE = 0164
DATA AREA = 004C
FREE SYMS = 0289
END COMPILATION

A>

```

Figure-5.5.6 コンパイル時に生成されたりスト・ファイルの一部をタイプアウト.

5.6 PL/M

5.6.1 PL/Mについて

PL/Mは、インテル社が1974年に発表した、8008および8080用のコンパイラであり、特にマイクロコンピュータのソフトウェアの開発に適するように、シンプルに作られたプログラミング言語です。

CP/Mの開発者で、デジタルリサーチ社の社長である Dr. Gary Kildall も、当時インテル社のコンサルタントとして、PL/Mの開発を行いました。

PL/Mは、最初はFORTRANで書かれ、その後アセンブラで書き直され、さらにその後、PL/M自身によって書き直されて現在に至っています。

PL/Mは、PL/Iに似た構造化プログラミング言語であり、マイクロプロセッサのきめ細かなコントロールがアセンブラに近いレベルで可能なマイクロコンピュータ・システムに密着した高級言語です。システムの記述が可能で、マイクロコンピュータの機能を十分に活用できる高級言語は、このPL/MとFORTHだけとも言われています。また、CP/Mのいくつかの部分は、PL/Mによって書かれています (version 2.0 になって、アセンブラで書き直された部分もあるが)。

5.6.2 Systems Consultants社 PLMXについて

PLMX は、インテル社の PL/M とコンパチブルのプログラミング言語であり、PL/M がインテル社の ISIS-II オペレーティング・システム専用であるのに対し、PLMX は、CP/M 上の PL/M と言っても良いでしょう。PLMX のマニュアルは、言語マニュアルが付属しておらず、そのユーザーズ・ガイドの中には、「PL/M に関しては、インテル社の『PL/M-80 Programming Manual (ドキュメント No.98-268B) を参照してくれ」と書かれているのがおもしろいところです。

PLMX コンパイラは、PL/M のソース・ファイルをコンパイルして、アセンブリ言語のソース・ファイルを出力します。よって、当コンパイラは、PL/M ソース・ファイルからアセンブリ言語のソース・ファイルへの「トランスレータ」と言ってもよいでしょう。出力されたアセンブリ言語のソース・ファイルの形式は、マイクロソフト社の MACRO-80 (3.3 章参照) に準じており、もちろんそのままアセンブルすることができますが、必要とあらば、そのソース・ファイルをエディタを使って、自由に書き替えて使用することもできます。

PLMX は、CP/M 上で実行するプログラムを開発する場合に便利のように、CP/M のシステム・コールとリンクした、ディスク・アクセスを含む I/O ユーティリティ・ライブラリを、エクスターナル (外部)・プロシジャとして用意しています。

5.6.3 PLMXによる「SAMPLE」プログラムの作成

まず、PL/M によるソース・プログラムを示します。

```
A>TYPE SAMPLE.SRC ..... ファイル名のエクステンションは何でもよいが、
                          "SRC"がデフォルト名に指定されている。
/* PLMX (PL/M) sample program for "Application CP/M" */...../*~*/の間がコメントとなる。

sample:
do;
  declare bufptr address;          /* setting buffer */
  declare status address;
  declare count address;
  declare buf1ptr address;
  declare temp address;
  declare numb address;
  declare msg1(15) byte data ('1+2+3+....n = x');
  declare msg2(28) byte data ('INPUT n (n = 1...250)');
  declare msg3(10) byte data ('1+2+3+....');
  declare msg4(3) byte data (' = ');
  declare crlf(2) byte data (0dh,0ah);
  declare buff1(10) byte;
  declare buff2(20) address;

/* system call procedure */
```

各種データの定義。


```

read:
procedure (function, buffer, count, actual, status) external;
declare (function, buffer, count, actual, status) address;
end read;

write:
procedure (function, buffer, count, status) external;
declare (function, buffer, count, status) address;
end write;

/* PL/M library call procedure */
numin:
procedure (buffer) address external;
declare buffer address;
end numin;
nmout:
procedure (value, base, lc, buffadr, width) external;
declare (value, buffadr) address;
declare (base, lc, width) byte;
end nmout;

/* main routine */

call write (0, .crlf, 2, .status); /* print message */
call write (0, .mssg1, 15, .status);
call write (0, .crlf, 2, .status);
call write (0, .mssg2, 28, .status);

/* read number */
call read (0, .buff1, 10, .count, .status); /* read */
buf1ptr = .buff1; /* set buffer pointer */
numb = numin (.buf1ptr); /* convert to binary */

do while numb > 250;
call write (0, .mssg2, 28, .status);
call read (0, .buff1, 10, .count, .status);
buf1ptr = .buff1;
numb = numin (.buf1ptr);
end;

call write (0, .crlf, 2, .status);
call write (0, .crlf, 2, .status); /* again */
call write (0, .mssg3, 10, .status); /* print message */
call nmout (numb, 10, ' ', .buff2, 4); /* binary to ascii */
call write (0, .buff2, 4, .status); /* write number */
call write (0, .mssg4, 3, .status); /* print '=' */

temp = numb; /* calculate */
do while numb >= 1;
numb = numb - 1;
temp = temp + numb; /* answer = temp */
end;

call nmout (temp, 10, ' ', .buff2(4), 7); /* binary to ascii */
call write (0, .buff2(4), 7, .status); /* write on console */
call write (0, .crlf, 2, .status); /* crlf at finish */

end sample;

A>

```

行入力, 行出力のライブラリを使用する手続き。

ASCII文字→16bit/バイナリ変換のライブラリを使用する手続き。

16bit/バイナリ→ASCII文字変換のライブラリを使用する手続き。

メインルチン

Figure-5.6.1 PL/M による "SAMPLE" プログラムのソース・ファイル。

ソース・ファイル名のエクステンションは何でも構いませんが、“SRC” がデフォルトとして指定されていますので、“SRC” としておく方が何かと便利です。

次に、PLMX のパッケージの中から “SAMPLE” プログラムを作成するために必要な最低限の各ファイルを示しておきます。

A>B:STAT *.*!

Recs	Bytes	Ext	Acc	
52	7k	1 R/W	A:PLMX.COM	コンパイラのルート・プログラム。
155	20k	2 R/O	A:CODA.PLM	
158	20k	2 R/O	A:FNLCG.PLM	
158	20k	2 R/O	A:PARSER.PLM	
41	6k	1 R/O	A:PLMLEX.PLM	コンパイラのオーバーレイ。 ("SAMPLE"プログラムでは使用されないものもある)
13	2k	1 R/O	A:IATABLE.FOR	
85	11k	1 R/O	A:IETABLE.FOR	
5	1k	1 R/O	A:IHTABLE.FOR	
84	11k	1 R/W	A:LBO.COM	リンク・ローダ。
157	20k	2 R/W	A:MBO.COM	マクロ・アセンブラ。
53	7k	1 R/W	A:IOLIB.REL	
29	4k	1 R/W	A:RLIB.REL	各種ライブラリ・ファイル。
22	3k	1 R/W	A:SAMPLE.SRC	"SAMPLE"のソース・ファイル。

Bytes Remaining On A: 109k

A>.

Figure-5.6.2 PLMX による “SAMPLE” プログラム開発に必要な最低限のファイル。

では、コンパイルの作業から始めます。コンパイラの実行と生成されたファイルの確認の実行例を次に示します。

A>DIR SAMPLE.*!最初の“SAMPLE”ファイルの確認。

A: SAMPLE SRCソース・ファイルのみ存在。

A>PLMX SAMPLE ; M+C+!コンパイラの実行。スイッチ“M+”は、当モジュールをメイン・モジュールに指定する。“C+”は、コンパイル・リストをコンソールに出力させる。

PLMX COMPILER VERSION 2.4

COPYRIGHT (C) 1980, SYSTEMS CONSULTANTS, INC.

TITLE SAMPL
NAME ('SAMPL')

SAMPL: EXTRN INIT@コンソールに出力されたコンパイル・リスト。

PUBLIC AAAAB@,AAAA@

AAAA@: LXI H,\$+6
JMP INIT@

AAAAB@:

;
;/* PLMX (PL/M) sample program for "Application CP/M" */
;


```

T0033:
        DS 02H
        END AAAAA@
END OF COMPILATION
000 ERROR(S) DETECTED

```

コンパイル成功。エラーなし。

A>DIR SAMPLE.* /生成されたファイルの確認。

A: SAMPLE SRC : SAMPLE MAC

ソース・ファイル

マイクロソフト社のMACRO-80形式のアセンブリ言語ソース・ファイル。
オブジェクト・ファイルではなく、ソース・ファイルであることに注目。

Figure-5.6.3 PLMX によるコンパイラの実行と生成されたファイルの確認。

ソース・ファイル "SAMPLE.SRC" のコンパイルは成功して、アセンブリ言語（マイクロソフト社の MACRO-80 形式）のソース・ファイルが生成されました。このことを「アセンブリ言語のソース・ファイルに展開された」とも表現します。

生成されたアセンブリ言語のソース・ファイル "SAMPLE.MAC" ("MAC" は、MACRO-80 のソース・ファイルに付けるエクステンション) を見てみましょう。その一部を次に示します。

A>TYPE SAMPLE.MAC /展開されたソース・ファイルをタイプアウト。

```

        TITLE SAMPL
        NAME ('SAMPL')
SAMPL::
        EXTRN INIT@
PUBLIC AAAAB@,AAAAA@

AAAAA@:
        LXI H,$+6
        JMP INIT@
AAAAB@:
;
; /* PLMX (PL/M) sample program for "Application CP/M" */
;
;
; declare (value, buffadr) address;
; declare (base, lc, width) byte;
; end nmout;
;
; /* main routine */
;
; call write (0, .CrLf, 2, .status); /* print message */
G0023:
G0020:
G001A:
G0013:
        LXI B,0H
        PUSH B

```

各種定義・宣言・手続きの部分。
こちら辺のものはコメント扱い
になっている。

メインルーチン部


```

        LXI B,A000F
        PUSH B
        LXI B,02H
        PUSH B
        LXI D,A0002
        POP B
        CALL WRITE
;
;   call write (0, .mssg1, 15, .status);
        LXI B,0H
        PUSH B
        LXI B,A0007
        PUSH B
        LXI B,0FH
        PUSH B
        LXI D,A0002
        POP B
        CALL WRITE
;
;   call write (0, .CrLf, 2, .status);
        LXI B,0H
        .
        .
T002A:
        DS 02H
T002B:
        DS 01H
T0033:
        DS 02H
        END AAAAA@

```

Figure-5.6.4 コンパイルにより生成されたアセンブリ言語のソース・ファイルのタイプアウト。

このソース・ファイルは、このままでアセンブルすることができる状態のものですが、特にアセンブリ言語のレベルで手を入れたいことがあれば、エディタを使ってこのソース・ファイルを自由に書き替えることができます。

次は、このソース・ファイルのアセンブルです。このままのソース・ファイル "SAMPLE.MAC" をMACRO-80 を使ってアセンブルし、それによって生成されるファイルを確認する例を示します。

A>M80 SAMPLE,SAMPLE=SAMPLE / -----MACRO-80によるアセンブルの実行。

No Fatal error(s)
アセンブル終了。エラーなし。

A>DIR SAMPLE.* / -----生成されたファイルの確認。

A:	SAMPLE	SRC :	SAMPLE	MAC :	SAMPLE	PRN :	SAMPLE	REL
PL/Mのソース・ファイル.		アセンブリ言語の		生成された		生成された		
A>		ソース・ファイル		リスト・ファイル		リロケータブル・オブジェクト・ファイル.		

Figure-5.6.5 コンパイルにより生成されたアセンブリ・ソース・ファイルを、MACRO-80 によりアセンブルする。

アセンブルが終了し、リロータブル・オブジェクト・ファイル ("REL" ファイル) とリスト・ファイル ("PRN" ファイル) が生成されています。

注) MACRO-80 と、LINK-80 については、3.3 章で取り上げていますので参照して下さい。

いよいよ次は最終ステップのリンク作業です。

PL/M のソース・プログラムの前半部に、外部ライブラリ・ルーチンを使用するための手続き文がありますが、そのライブラリを含む2つのオブジェクト・ファイル "RLIB. REL" と "IOLIB. REL" を、メインルーチンの "SAMPLE. REL" にリンクします。

その実行例と生成されたファイルの確認を次に示します。

これらをリンクして、"SAMPLE" というファイル名の "COM" ファイルを作成する。

メイン・モジュール ライブラリ1 ライブラリ2 リンク作業終了と同時に CP/M に戻る。
"COM" ファイルを作成。

```
A>L80 SAMPLE,RLIB,IOLIB/S,SAMPLE/E/N)
```

LINK-80 3.44 09-DEC-81 COPYRIGHT (C) 1981 MICROSOFT
%MULT. DEF. GLOBAL BP67@ このエラー・メッセージは、BP67@ が、RLIB と IOLIB の双方に存在する
ためのマルチ・ディファイン・エラー。無視してよい。

```
DATA 0103 0B56 < 2643>
```

29308 BYTES FREE 11 ページ (256 バイト × 11) の純マシン・コードができた。
[0146 0B56 11]

リンク終了。

A>DIR SAMPLE.* 生成されたファイルの確認。

```
A: SAMPLE SRC : SAMPLE MAC : SAMPLE PRN : SAMPLE REL
A: SAMPLE COM
```

実行可能な純マシン・コード・ファイル

A>B:STAT SAMPLE.COM "SAMPLE.COM" のファイル容量を調べる。

```
RECS BYTES EXT ACC
21 3K 1 R/W A: SAMPLE.COM      3K バイト長である。
BYTES REMAINING ON A: 72K      (本章の言語の中では、Rgy FORTH に次いでコンパクトである)
```

A>

Figure-5.6.6 アセンブルにより生成されたメイン・モジュールとライブラリ・モジュールとのリンク。

最終目的である実行可能な純マシン・コードのファイルができ上がりました。サイズは3Kバイトで、他の言語に比べるとかなりコンパクトであり、本章の中では、Rgy FORTH に次いで小さいものです。では、でき上った "SAMPLE" プログラムを実行してみます。

A>SAMPLE / "SAMPLE"プログラムの実行.

```

1+2+3+...n = x
INPUT n (n = 1...250)    -->260! .....261以上の入力値エラー。
INPUT n (n = 1...250)    -->100! .....再度正しい値を入力。

1+2+3+... 100 =      5050 .....入力値と答えが出力されている。

```

Figure-5.6.7 PLMX (PL/M) で作成された "SAMPLE" プログラムの実行例.

参考までに、MACRO-80 でのアSEMBル時に生成された、リスト・ファイル "SAMPLE.PRN" をタイプアウトして、その一部を示しておきます。

A>TYPE SAMPLE.PRN /アセンブルにより生成された“PRN”ファイルのタイプアウト。

SAMPL MACRO-B0 3.44 09-Dec-81
PAGE 1

0000' ← "は相対アドレスを表す。

0000' 21 0006'

0003' C3 0000*

0006' ← "*"は外部参照を表す。

0006'

0006'

0006'

0006'

0006' 01 0000

0009' C5

000A' 01 01D5'

000D' C5

000E' 01 0002

0011' C5

0012' 11 0002"

0015' C1

0016' CD 0000*

0019' 01 0000

```

TITLE SAMPL
NAME ('SAMPL')

SAMPL:
    EXTRN INIT@
PUBLIC AAAAB@,AAAAA@

AAAAA@:
    LXI H,*+6
    JMP INIT@

AAAAB@:
;
; /* PLMX (PL/M) sample program for "Application CP/M" */
;

; /* main routine */
;
;    call write (0, .crlf, 2, .status); /* print message */
G0023:
G0020:
G001A:
G0013:
    LXI B,0H
    PUSH B
    LXI B,A000F
    PUSH B
    LXI B,02H
    PUSH B
    LXI D,A0002
    POP B
    CALL WRITE
;
;    call write (0, .msg1, 15, .status);
    LXI B,0H

```

Figure-5.6.8 M80 によるアセンブルで生成された PRN ファイルのタイプアウト.

PLMX の、その他の多くの機能の紹介は省略します。

5.7 C

5.7.1 Cについて

Cは、1971年に稼動を始めた UNIX オペレーティング・システムで使用するメインの言語として開発されました。

“UNIX” というのは、ベル研究所の Dennis Ritchie らによって設計され、当初は、DEC のPDP-11 上にインプリメントされた OS であり、これからの16ビット、32ビットコンピュータの共通 OS の1つとして、注目され広まりつつあります（注：UNIX は、CP/M や MS-DOS の延長線に位置するものではなく、明らかに使用環境が異なるものであり、両者を比較するのは意味がありません）。

“C” は、1970年に K. Thompson によって開発された “B” 言語と関係が深く、“C” という名も “next B” という意味あいであ付けられたとされています。

UNIX は、1971年当初はアセンブラで書かれていましたが、1973年に全面的に自分自身の “C” によって書き直されています。

Cが使われた身近な例では、多くのパーソナル・コンピュータに採用されているおなじみのマイクロソフト BASIC が、アセンブラ記述であったものを現在、Cによって書き直されていると聞いています。また、同社の世界最高レベルの簡易言語 “Multiplan” はCで書かれています。

これらの事実からも分かるように、Cはシステムを記述したり、言語を記述したり、事務用プログラムを記述したりすることができる言語であり、さらに、アセンブラに近い記述をすることも可能であるため、非常に幅広い分野で利用することができます。

C言語の特徴の1つに、ソース・プログラムをトップ・ダウン（上から順に、メインルーチン→サブルーチン→サブルーチンのサブルーチン→そのまたサブルーチン→……という具合に書きおろしていくこと）で記述できることも挙げられます。

5.7.2 BD Software社 C Compilerについて

現在、入手が可能なC言語は、Whitesmiths 社のものと、Super Soft 社のものと、それにこの BD Software 社のものが一般的ですが、3者の中では、この BDS 社のものが何と言っても使い易いので取り上げてみました。

BDS 社のCコンパイラを構成している主要なモジュールは、

CC1.COM ……フェーズ1の段階のコンパイラ。シンボル・テーブルとエンコードされたソース・コードを生成し、メモリ上あるいはディスク上に置く、コンパイル・エラーがなければ自動的に CC2 に進む。

CC2.COM ……フェーズ2のコンパイラ。CC1で生成された“CC1”ファイルを入力して、“CRL”(C ReLocatable)ファイルを生成、リロケートブルのオブジェクト・ファイルができ上る。

CLINK.COM… “CRL”ファイルとC.CCC(コモン・システム・サブルーチン)ファイルをリンクし、実行可能な“COM”ファイルを生成する。

CLIB.COM……CC2で生成された“CRL”ファイルに対して、それに含まれる各種ファンクションを、“CRL”ファイル間相互で移動などの操作を行うライブラリアン(後述)。

C.CCC……………ランタイムの基本ファイル。コマンド・ライン処理、ファイル入出力バッファや、多くのサブルーチンなどで構成されている。CC2で生成された“CRL”ファイルとリンクされ、独立して実行可能な“COM”ファイルに組み込まれる。

DEFF.CRL……標準的な各種ライブラリ。CLINKによって、メイン“CRL”ファイルに組み込まれる。

以上の4つの“COM”ファイルと、標準ライブラリそれにラン・タイム・モジュールがあり、その他には、標準ライブラリなどのソース・ファイル、各種ユーティリティやゲームなどのソース・ファイルが含まれています。

BDSのCコンパイラの出力は、Super Soft社のCコンパイラのように、アセンブリ言語のソース・コードではないので、コンパイル後、生成されたアセンブリ言語のソース・コードに手を入れて、細かな操作をするという訳には行きません。

しかし、その反面、大変使い易く、C言語を学びながら実務にも応用しようとする人には最適ではないかと思います。

このC言語は、UNIX上のCを基に、重要でない機能を省き、マイクロコンピュータ上のCとして構成された、UNIX Cのサブセット・レベルのものです。

5.7.3 BDS Cによる「SAMPLE」プログラムの作成

まず、Cによる“SAMPLE”プログラムのソース・ファイルを示します。ファイル名のエクステンションは、必ず“C”でなければなりません。

A>TYPE SAMPLE.C ……C言語のソース・プログラムをタイプアウト。エクステンションは必ず“C”とする。

```
/* BD Software C Compiler sample program for "Application CP/M" */
/*~~~~*/の間がコメントとなる。
```



```

main()                                /* main routine */
{
    int i,num,temp;

    printf ("n1+2+3+...n = x"); /* print msg */
    printf ("ninput n (n = 1...250) -->");
                                   /* "※"記号は本来は/バックスラッシュ/ である。
                                   日本のプリンタでは"※"になってしまう。
                                   ※nは改行を表す。 */

    num = getnumber();              /* get number */

    while (num > 250) { /* 複数の文を!!で囲むと1つのブロックにすることができる。 */
        printf("input n (n = 1...250) -->");
        num = getnumber(); }

    printf ("n1+2+3+...");          /* print msg */
    printf ("%3d",num);              /* print number */
    printf (" = ");

    temp=num;                       /* calculate */
    while (--num) {
        temp = temp+num; }

    printf ("%6d",temp);             /* print answer */
    printf ("%n");                  /* cr on screen */
}

getnumber()                         /* get number subroutine */
{
    int p,nmb;
    char numb,s[51];
                                   /* このようにサブルーチンを、
                                   それが使われた後で定義する
                                   ことができる。 */

    p = 0;
    while (numb != '\n') {          /* looking 'cr' */
        numb = getchar();
        s[p] = numb;
        ++p; }

    p = 0;                          /* convert to integer number */
    nmb = 0;
    while (s[p] != '\n' && s[p] >= '0' && s[p] <= '9' ) {
        nmb = 10 * nmb + s[p] - '0';
        ++p; }

    return (nmb);                  /* return with integer number */
}
A>

```

Figure-5.7.1 Cによる“SAMPLE”プログラムのソース・ファイル。

次に、BDSのCコンパイラのパッケージの中から、“SAMPLE”プログラムを作成するのに必要な最低限の各ファイルを示します。


```
A>B:STAT *.* /
```

Recs	Bytes	Ext	Acc	
86	11k	1	R/O	A:CC1.COM フェーズ1のコンパイラ.
70	9k	1	R/O	A:CC2.COM フェーズ2のコンパイラ.
26	4k	1	R/W	A:CLINK.COM リンク・ローダ.
14	2k	1	R/W	A:C.CCC 基本ランタイム・モジュール.
77	10k	1	R/W	A:DEFF.CRL 標準ライブラリ.
10	2k	1	R/W	A:SAMPLE.C "SAMPLE"のソース・プログラム.

Bytes Remaining On A: 203k

A>

Figure-5.7.2 BDS のCによる "SAMPLE" プログラムの開発に必要な最低限の各ファイル.

では、コンパイルの作業から始めます.

BDS のCは、実行可能な純マシン・コード・ファイルを作成するまでの手順が非常にシンプルであるのが特徴の1つです.

コンパイラの実行と、生成されたファイルの確認例を次に示します.

```
A>DIR SAMPLE.* / .....最初の"SAMPLE"ファイルの確認.
```

```
A: SAMPLE C .....ソース・ファイルのみ存在.
```

```
A>CC1 SAMPLE.C / .....フェーズ1のコンパイラの実行.
```

```
BD Software C Compiler v1.43a (part I)
25K unused .....フェーズ1のコンパイル完了.
```

```
BD Software C Compiler v1.43 (part II) .....自動的にフェーズ2のコンパイラ(CC2.COM)
20K to spare .....が起動し、実行される.
```

コンパイル成功. エラーなし.

```
A>DIR SAMPLE.* / .....生成されたファイルの確認.
```

```
A: SAMPLE C : SAMPLE CRL
ソース・ファイル .....生成されたリロケータブル・オブジェクト・ファイル.
```

A>

Figure-5.7.3 BDS のCによるコンパイラの実行と生成されたファイルの確認.

"CC1.COM" によるフェーズ1のコンパイルが終了した時点で、自動的に "CC2.COM" によるフェーズ2のコンパイラが起動され実行されました(スイッチにより、別々に実行することも可能です).

これにより、リロケータブルのオブジェクト・ファイル "SAMPLE.CRL" が生成されています.

次はリンク・ローダの実行です. "SAMPLE.CRL" に, "C.CCC" と "DEFF.CRL" をリンクします. その実行例を次に示します.

A>CLINK SAMPLE /リンク・ローダの実行。"SAMPLE"に、"C, CCC"と"DEFF, CRL"をリンクする。

BD Software C Linker v1.43

Linkage complete

32K left over

リンク完了。

A>DIR SAMPLE.* /生成されたファイルを確認する。

A: SAMPLE C : SAMPLE CRL : SAMPLE COM

生成された実行可能な
純マシン・コード・ファイル

A>B:STAT SAMPLE.COM / "SAMPLE.COM"のファイル容量を調べる。

Recs Bytes Ext Acc

29 4k 1 R/W

A: SAMPLE.COM4K/バイト長である。

Bytes Remaining On A: 30k

A>

Figure-5.7.4 リンク・ローダの実行。"SAMPLE.CRL" に、ライブラリをリンクする。

以上の作業で、BDS のCによる実行可能な純マシン・コードのファイルができ上がりました。ではその "SAMPLE" プログラムを実行してみましょう。

A>SAMPLE / "SAMPLE"プログラムの実行。

1+2+3+....n = x

input n (n = 1...250) -->350 /251以上なので入力値エラー。

input n (n = 1...250) -->200 /今回は正しい値を入力。

1+2+3+....200 = 20100入力値と答えが出力されている。

A>

Figure-5.7.5 BDS のCにより作成された "SAMPLE" プログラムの実行。

BDS のCの、その他の多くの機能の紹介は省略します。

5.8 FORTH

5.8.1 FORTHについて

FORTH は、1969年に、米国バージニア州の国立電波天文台で、観測装置の自動化を行うための適当なプログラミング言語がなかったため、Charles H. Moore によって新たに開発された言語です。

Moore はその後、FORTH 社を設立し、ミニコンピュータやマイクロコンピュータ用の Poly FORTH などの製品を発表しています。

FORTH は、本章で取り上げた他のコンパイラ言語とは、文法やオブジェクト構造が大きく異なり、“ALGOL の流れをくむ”とか、“PL/I と PASCAL の中間的なもの”と言ったような表現ができる種類のものではありません。

FORTH は、スタック言語と言ってもよく、すべての数値処理はスタックに積むことにより行われます（スタックとは、アセンブリ言語の“PUSH”、“POP”と同じ原理です）。また、スタック言語であるため、その文法は必然的に逆ポーランド記法となっています（“2 + 3”を逆ポーランド記法では“2 3 +”となる）。

FORTH の記述法は、構造化プログラミングの最たるものであり、すべてがモジュール定義の形をとり、メインルーチン、サブルーチン、関数、演算子といった区別はありません。

FORTH は、システムが記述ができ、かつアセンブラに近い記述が可能で、制御用プログラミングにも適した非常にオブジェクト効率のよいコンパイラとして注目されています。

5.8.2 Rgy FORTHについて

^{リギー}Rgy FORTH は、Z80バージョンを「FZ80」、8080バージョンを「F80」と呼んでいます。ここではZ80バージョンのFZ80を使用します。

Rgy FORTH は、純国産のFORTH 言語であり、現在入手可能な他のFORTH にはない、いくつかの特徴を持っています。

- FORTH 言語と言うより、FORTH 言語も記述できるアセンブラと言ってもよく、FORTH 言語を使わなければ、全くのアセンブラとして動作する。
- 現時点では、おそらく最速の FORTH である。
- アドレス／オブジェクトを含んだ完全なコンパイル・リストを出力する。
- FORTH ワード中からのアセンブラ・プログラムの呼び出し、またはその逆の呼び出しを任意のタイミングで何回でも行うことが可能。
- 割り込み処理を FORTH レベルで記述できる。
- CP/M の DDT とコマンド・コンパチで、コンパイル後のオブジェクトを、シンボリックに対話形式でデバッグできるデバグが、“FDT”が付属している。

など、この Rgy FORTH の開発者が従来の FORTH に満足せず、さらにマイクロ・プロセッサに密着した、アセンブラに替り得る高級言語をターゲットとしていることがよく表れています。

PL/M の項でも述べましたが、マイクロプロセッサに密着したプログラミング言語で、オブジェクトの小さいことや複雑な割り込み処理その他で、現時点で PL/M に対抗できるのは、この Rgy FORTH

がその筆頭と言えるでしょう。

5.8.3 Rgy FORTHによる「SAMPLE」プログラムの作成

まず、Rgy FORTH による "SAMPLE" プログラムのソース・ファイルを示します。ファイル名のエクステンションは、必ず "F80" でなければなりません。

A>TYPE SAMPLE.F80ファイル名のエクステンションは、必ず"F80"とする。

※ Rgy FORTH sample program for "Application CP/M" ※※は本来はバックスラッシュ、
"※"からラインの終わりまではコメントとなる。※は本来はバックスラッシュ、
"※"であるが、日本のプリンタ
では"※"となってしまう。

TITLE 'SAMPLE PROGRAM'

READLIBカーネル(核)とリンクする擬似命令。

: [?0-9] DUP 2FH > SWAP 3AH < FAND ;入力コードが0~9であるかをチェックするワード。

: [ASCII-BIN]

※ENTRY TOP:ASCII NIBL

※EXIT TOP:1/0(1=OK)

※2ND: BINARY DATA

DUP ?0-9

IF '0' - , 1

ELSE DROP, 0

THEN ;

1桁10進ASCII→バイナリ変換のワード。

CONBUFF DS 4 + 2

: [STRING>NUMBER]

※EXIT TOP:1/0

※2ND: BINARY DATA

(CONBUFF + 2) >R

0

BEGIN

I B@ ASCII-BIN

IF SWAP 10 * +

R> 1+ >R

REPEAT

R> B@ 0=文字列の最後がCRであるかのチェック。

IF 2DUP 0 > SWAP 251 < FAND

IF 1

ELSE DROP 0

THEN

ELSE DROP 0

THEN ;

1~250の範囲
かをチェック。

ASCII文字列を10進文字として、
これをバイナリ値に変換するワード。

MSG_INPUT:

DB 'input n (n = 1...250) -->%00'

: [INPUT_NUMBER]

※EXIT TOP:1/0

※ 2ND: BINARY DATA

BEGIN

PRCRLF , MSG_INPUT PRSCR/LFとメッセージの出力。

4 CONBUFF RD_CONBUFFコンソールから1桁入力。

IF STRING>NUMBER

ELSE 0

THEN

UNTIL ;

BASIC言語での「INPUT」に
相当するワード。

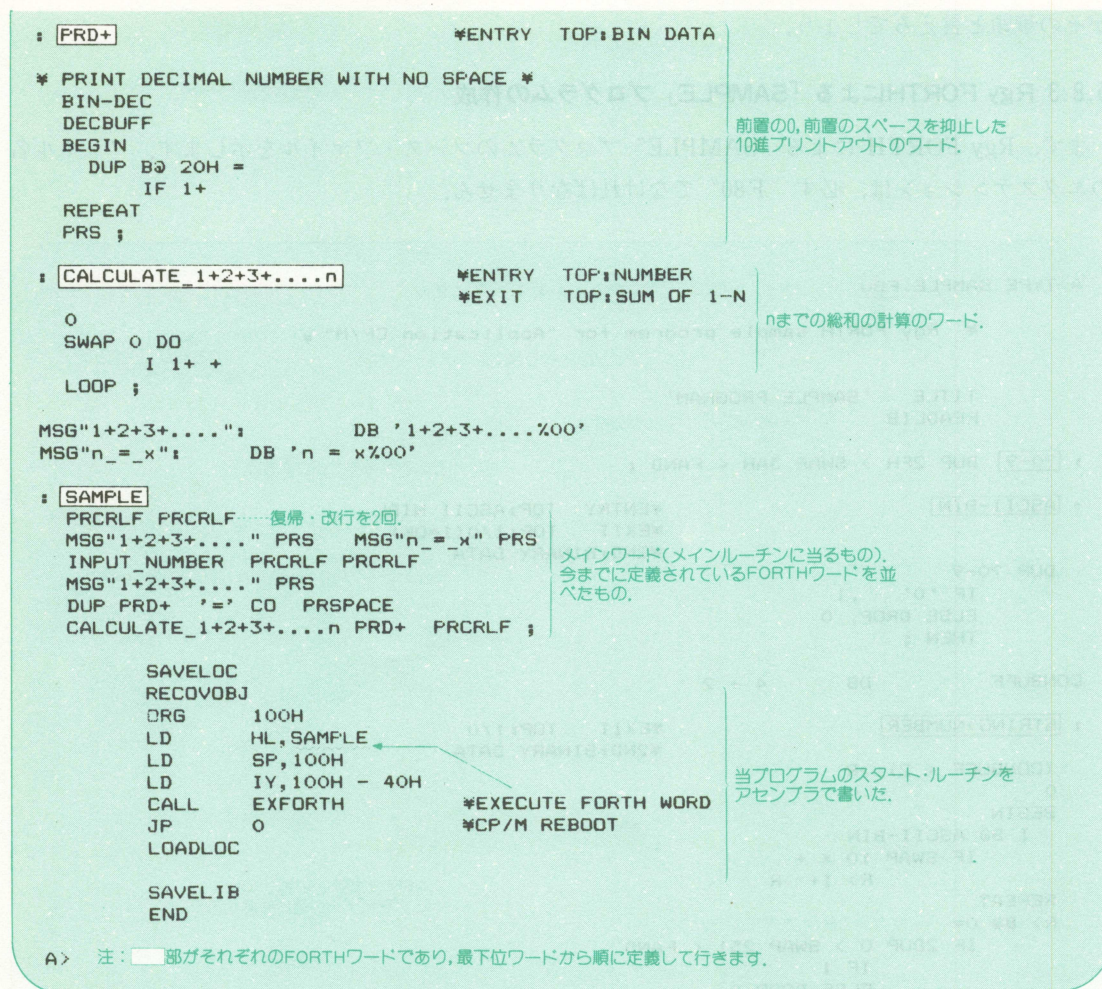


Figure-5.8.1 Rgy FORTH による "SAMPLE" プログラムのソース・ファイル。

このように FORTH の文法は, COBOL や PL/I などとは全く雰囲気が異なっています。

Rgy FORTH のパッケージの中から, "SAMPLE" プログラムを開発するのに必要な最低限の各ファイルを次に示しておきます。

この中の "CPMCON.HEX" は, あらかじめそのソース・ファイル "CPMCON.F80" から作成しておきます。


```
A>B:STAT *.*]
```

```

Recs  Bytes  Ext  Acc
142    18k    1  R/W  A:FZ80.COM  コンパイラ.
44      6k    1  R/W  A:CPMCON.HEX  CP/Mコンソール/ライブラリ.
60      8k    1  R/W  A:FDT.COM  テンプレート(直接は必要ない).
12      2k    1  R/W  A:SAMPLE.F80  "SAMPLE"のソース・ファイル.
Bytes Remaining On A: 266k

```

```
A>
```

Figure-5.8.2 Rgy FORTH による "SAMPLE" プログラムの開発に必要な最低限の各ファイル。

さてコンパイルの作業を始めますが、Rgy FORTH はコンパイル時に各種ライブラリも同時にリンクできるので、この場合は、CP/M のコンソール入出力ライブラリ "CPMCON . HEX" をコンパイル時にリンクします。

また、ソース・プログラムの最下部で、当プログラムの起動ルーチンが定義されており（プログラムの起動ルーチンまで定義できることは、他のコンパイラにはない特徴の1つ）、コンパイラの出力は、すでに絶対アドレスを持ったオブジェクト・コードとなっています。

コンパイラの実行例を次に示します。

```
A>DIR SAMPLE.*] ----- 最初の"SAMPLE"ファイルの確認.
```

```
A: SAMPLE F80 ----- ソース・ファイルのみ存在.
```

```
      ↓
      コンパイルと同時にリンクするライブラリ.
```

```
A>FZ80 SAMPLE CPMCON ] ----- コンパイラの実行. コンパイルと同時に、CP/Mコンソール
      | ライブラリの"CPMCON.HEX"をリンクする.
```

```

RIGY CO.      Z80 FORTH COMPILER "FZ80 V2.0"      18205159
PASS1.END
PASS2.END
NEXTCP=07ABH  NEXTDP=07ABH  FREESYMN0= 1366
COMPLETE COMPILATION

```

コンパイル成功. エラーなし.

```
A>DIR SAMPLE.*] ----- 生成されたファイルの確認.
```

```

A: SAMPLE F80 : SAMPLE HEX : SAMPLE PRN : SAMPLE ERR
  ソース・ファイル   生成されたインテルHEX形式の  生成されたリスト  エラー・インフォメーション・ファイル
                     オブジェクト・ファイル      ファイル          (この場合は、エラーなしなので中身は空)
A>

```

Figure-5.8.3 Rgy FORTH のコンパイラの実行と生成されたファイルの確認。

コンパイルは成功して、HEX 形式のオブジェクト・ファイルが生成されています。このファイルは CP/M の DDT や Rgy FORTH の FDT など、メモリ上に実行可能な純マシン・コードとしてロードでき、また、CP/M の LOAD コマンドで実行可能な "COM" ファイルに変換して、ディスクにセーブすることができます。

次に、CP/M の LOAD コマンドの実行と生成されたファイルの確認例を示します。

```
A>B:LOAD SAMPLE ] ..... CP/MのLOADコマンド実行、"SAMPLE.HEX"から"SAMPLE.COM"を生成する。

FIRST ADDRESS 0100
LAST ADDRESS 07A7
BYTES READ 069A
RECORDS WRITTEN 0E

LOADコマンド終了。
A>DIR SAMPLE.* ] ..... 生成されたファイルの確認。

A: SAMPLE F80 : SAMPLE HEX : SAMPLE PRN : SAMPLE ERR
A: SAMPLE COM
..... 実行可能な純マシン・コードのファイルが生成された。

A>B:STAT SAMPLE.COM ] ..... "SAMPLE.COM"のファイル容量を調べる。

Recs Bytes Ext Acc
14 2k 1 R/W A:SAMPLE.COM ..... 2Kバイト長。本章で取り上げたコンパイラの中で最もコンパクト。
Bytes Remaining On A: 246k

A>
```

Figure-5.8.4 コンパイルにより生成された“HEX”ファイルに対し、LOAD コマンドの実行。

このように他のコンパイラと比較して、大変コンパクトなオブジェクト・ファイルができ上がりました。本章で取り上げた言語の中では最小です。

でき上がった“SAMPLE”プログラムを実行してみましょう。

```
A>SAMPLE ] ..... "SAMPLE"プログラムの実行。

1+2+3+....n = x
input n (n = 1...250) -->300 ] ..... 251以上の入力値のエラー。
input n (n = 1...250) -->250 ] ..... 再度正しい値を入力。

1+2+3+....250= 31375 ..... 入力値と答えが出力されている。

A>
```

Figure-5.8.5 Rgy FORTH により作成された“SAMPLE”プログラムの実行。

次に参考までに、Rgy FORTH に付属のデバグガである FDT の実行例と、コンパイルにより生成されたリスト・ファイル“SAMPLE.PRN”の一部を示しておきます。

→印は入力する行。
 シンボリックにデバッグが可能となるように、シンボル・テーブルを読み込む。
 A>FDT SAMPLE.HEX S] "SAMPLE.HEX"に対してFDTを起動。
 RIGY CO. Z80 FORTH DEBUGGER "FDT(Z80) V2.0"
 0100-07A7 NEXT=07AB PC=0000 SAVE= 7 FREEMAX=A3D8 (SYMBOL IN)
 →*F FORTHワードを実行するコマンド。
 →F '8' [?]0-9 ASCIIの0~9をチェックする。入力として"8"を与えた。
 FORTH RETURN
 PS(AEFE) 0001 リターンされた値は"真"である。
 →F 'A' ?0-9 今回は"A"を与えた。
 FORTH RETURN
 PS(AEFE) 0000 リターンされた値は"偽"である。
 →F '5' [ASCII-BIN] ASCII→バイナリ変換のワード、入力として"5"を与えた。
 FORTH RETURN
 →PS(AEFC) 0005 0001
 →F ^C 変換OK。
 A> 変換された値。
 Ctrl-CでCP/Mに戻る。

Figure-5.8.6 Rgy FORTH デバッガ FDT の実行。

```

A>TYPE SAMPLE.PRN ]
RIGY CO.      Z80 FORTH COMPILER "FZ80 V2.0"      18205159      PAGE      1

                                * Rgy FORTH sample program for "Applic
                                ation CP/M" *

                                TITLE  'SAMPLE PROGRAM'
                                READLIB

065A:E9DD 0237 015C 002F 0351.. : ?0-9  DUP 2FH > SWAP 3AH < FAND ;

0670:E9DD                                : ASCII-BIN      *ENTRY TOP:ASC
                                II NIBL      '
                                (1=OK)      *EXIT TOP:1/0
                                A            *2ND: BINARY DAT
                                DUP ?0-9
0672:0237 065A                        IF '0' - ,1
0676:0171 0000 015C 0030 02D6..      ELSE DROP, 0
0682:017A 0000 0253 016B              THEN ;
068A:013E

068C                                CONBUFF      DS      4 + 2

0692:E9DD                                : STRING>NUMBER      *EXIT TOP:1/0
                                A            *2ND: BINARY DAT
0694:015C 068E 0295                  (CONBUFF + 2) >R

```



```

      .
RIGY CO.      Z80 FORTH COMPILER "FZ80 V2.0"      18205159      PAGE      3
SAMPLE PROGRAM

**** SYMBOL-TABLE ****

!          F 01D2      $$+LOOP      F 01C0      $$-IF      F 0182
$$COLON    F E9DD      $$DO          F 0189      $$ELSE      F 017A
$$FALSE    F 016B      $$IF          F 0171      $$JMP        F 017A
$$LITERAL  F 015C      $$LOOP        F 019E      $$REPEAT     F 017A
$$SEMI     F 013E      $$TRUE        F 0165      $$UNTIL     F 0171
*          F 02F9      +              F 02CF      +!          F 0206
      .
      .
      .
      .
SW!        F 021E      SWAB          F 03CC      SWAN          F 03D3
SWAP       F 025A

NEXTPC=07A8H  NEXTDP=07A8H  FREESYMNO= 1366
COMPLETE COMPILATION

A>

```

Figure-5.8.7 リスト・ファイルのタイプアウト。

Rgy FORTH の、その他の多くの機能の紹介は省略します。

5.9 LISP

5.9.1 LISPについて

LISP は、1960年に出版された John McCarthy による、「Recursive Functions of Symbolic Expressions and Their Computation by Machine」という著書がその基本となっており、同年には最初の LISP がすでに稼動しています。この LISP は現在では、純 LISP と呼ばれています。

LISP の特徴は、第1にリスト処理です。そのリスト構造は、COBOL や、FORTRAN、PL/I などでは書き表せない、実行中に動的に変化するデータ構造を再帰的に定義することができるものです。

LISP は、数式の記号的な処理、コンピュータによるパズルの解読や定理証明などの人工知能研究の分野に、なくてはならないプログラミング言語として広く使われています。

LISP は、整数、実数、文字数、配列などのデータの型の区別がない単一データ形式であり、また、LISP のプログラムを LISP のデータとして処理して、プログラムとして実行するというようなことも可能です。

LISP は、一度作られたデータを書き換えることはなく（純 LISP）、書き換えが必要な場合は、旧

データはそのままにして、新データを新しく作って使用します(LISP 1.5には、書き換えの機能あり)。そのため、メモリをどんどん消費してしまうので、“Garbage Collect: ガーベジ・コレクト”と呼ばれる不用データの“ゴミそうじ”の機能を持ったシステム・サブルーチンが用意されているのも特徴です。

いずれにしても LISP は、一般的な言語と相当に趣を異にするものであり、COBOL や FORTRAN のプロフェッショナルと言えども、最初しばらくはとまどうことになるでしょう。

現在、LISP と言えば普通、LISP 1.5 (純 LISP に諸機能を追加したもの) を指します。

5.9.2 The Soft Warehouse社 muLISPについて

muLISP は、LISP 1.5にいくつかの機能を追加した LISP 1.5の上位コンパチブルに当たる LISP です。

muLISP の最初のバージョンは、1977年に発売された muLISP-77であり、その後1979年と1980年に大きなバージョン・アップが行われ、現在は、muLISP-80となっています。ここで取り上げたのは muLISP-80であり、これは従来の muLISP に、さらに多くのコードと高速処理を実現するために、疑似コード・コンパイラとインタープリタを追加したものです。muLISP の特徴は、

- 合計85の LISP ファンクションが、マシン語で定義されている。
- 2パスのコンパクトなガーベジ・コレクタが用意されており、データ・スペースのすべてのメモリをダイナミックに管理し、さらにガーベジ・コレクションの実行後、自動的にデータのリロケーションが最適となるようにダイナミックなメモリ操作が行われる。
- エディタとトレースの会話形レジデント・デバグが付属している。
- ファンクション定義は、自動的にDコード (distilled-code) にコンパイルされ、メモリの節約と高速化を可能としている。

などであり、muLISP は中間コード (Dコード) にコンパイルした後、インタープリタで実行するという形式をとっています。

5.9.3 muLISPによる「SAMPLE」プログラムの作成

まず、muLISP による “SAMPLE” プログラムのソース・ファイルを示します。ファイル名のエクステンションは何であっても実行には関係ありませんが、“LIB” がデフォルトに指定されているので、これに従います。


```

A>TYPE SAMPLE.LIB ..... ファイル名のエクステンションは何であってよいが、
                          "LIB"がデフォルトに指定されている。
      % muLISP sample program for "Application CP/M" %
                          %~%の間がコメントとなる。

(PROG1 ""                                     % Definition of DEFUN %
  (PUTD DEFUN (QUOTE
    (NLAMBDA (FUNC DEF)
      (PUTD FUNC DEF)
      ""))
  )

[DEFUN SAMPLE (LAMBDA (N NUMB TEMP)
  (TERPRI)                                     % CR & LF %
  (PRINT "1+2+3+....n = x")                   % title %

  (LOOP (PRIN1 "input n (n = 1...250) -->") % prompt message %
    (SETQ N (READ))                            % input n %
    ((AND (NUMBERP N)                          % is number ? %
      (PLUSP N)                                % n > 0 ? %
      (LESSP N 251))))                        % n < 256 ? %

    (SETQ TEMP (SETQ NUMB N))                  % TEMP := NUMB := n %
    (LOOP (SETQ NUMB (DIFFERENCE NUMB 1))      % NUMB := NUMB - 1 %
      ((ZEROP NUMB))                          % if NUMB = 0 then exit %
      (SETQ TEMP (PLUS TEMP NUMB)))           % TEMP := TEMP + NUMB
                                          and repeat %

  (TERPRI)
  (PRIN1 "1+2+3+.... ") (PRIN1 N)             % print message and N %
  (PRIN1 " = ") (PRINT TEMP)                  % print answer %
  (TERPRI)
  (SYSTEM) ..... 最後にCP/Mに戻るためのファンクション。 % return to CP/M %

[SAMPLE (RDS) ..... "SAMPLE"の実行。          % execute program %

```

関数"DEFUN"の定義.

"SAMPLE"の定義.

Figure-5.9.1 muLISP による "SAMPLE" プログラムのソース・ファイル.

次に, muLISP のパッケージの中から, "SAMPLE" プログラムを開発するために必要な最低限のファイルを示します.

```

A>B:STAT *.* /

```

RECS	BYTES	EXT	ACC
80	10K	1 R/W	A: MULISP.COM コンパイラ/インタープリタ.
125	16K	1 R/W	A: MUSTAR.SYS スクリーン・エディタ デバッグ. (直接には必要ない)
11	2K	1 R/W	A: SAMPLE.LIB "SAMPLE"プログラムのソース・ファイル.

BYTES REMAINING ON A: 213K

A>

Figure-5.9.2 muLISP による "SAMPLE" プログラムの開発に必要な最低限のファイル.

muLISP は、中間言語のインタプリタなので、まずインタプリタ（中間言語へのコンパイラを含む）をロードして、その上でソース・ファイルを読み込んで、そのままプログラムの実行に移ります。よって、コンパイルやリンクの作業は必要ありません。

"SAMPLE" プログラムの実行例を次に示します。

```
A>MULISP / .....インタプリタ(中間言語へのコンパイラを含む)を起動.

muLISP-80 2.15 (03/01/82)
Standard CP/M Version
Copyright (C) 1981 The SOFT WAREHOUSE
Licensed by MICROSOFT, Inc.

$ (RDS SAMPLE LIB) .....インタプリタが起動すると、プロンプトの"$"が出力される。
SAMPLE .....RDS(redirection)で、入力をコンソールから"SAMPLE. LIB"ファイルに変換する。
                                     ファイル名の"."を書いてはいけない。

$ | この間に、自動的にソース・ファ
$ | イルがロードされ、中間言語にコ
$ | ンパイルされ、インタプリタに
$ | による実行が開始される。

1+2+3+....n = x
input n (n = 1...250)    -->280 / .....251以上の入力値エラー。
input n (n = 1...250)    -->100 / .....今回は正しい入力。

1+2+3+.... 100 = 5050 .....入力値と答えが出力された。

A>
```

Figure-5.9.3 muLISP による "SAMPLE" プログラムの実行例。

muLISP には、LISP プログラムの開発のために、muSTAR と呼ぶ Artificial Intelligence Development System(AIDS)が付属しています。このツールがFigure-5.9.2の"muSTAR. COM"であり、この AIDS により LISP プログラムの作成に適したスクリーン・エディタとトレース機能を持ったデバugg、それにディスク・ファイルの操作などが可能となります。

参考までに、muSTAR を起動して、最初のメニュー画面を示しておきます。

```
A>MULISP MUSTAR / .....muLISPからmuSTARを起動.

muLISP-80 2.15 (03/01/82)
Standard CP/M Version
Copyright (C) 1981 The SOFT WAREHOUSE
Licensed by MICROSOFT, Inc.

この時点でスクリーンがクリアされ、次のコマンドメニューが出力される。

O P T I O N S
```

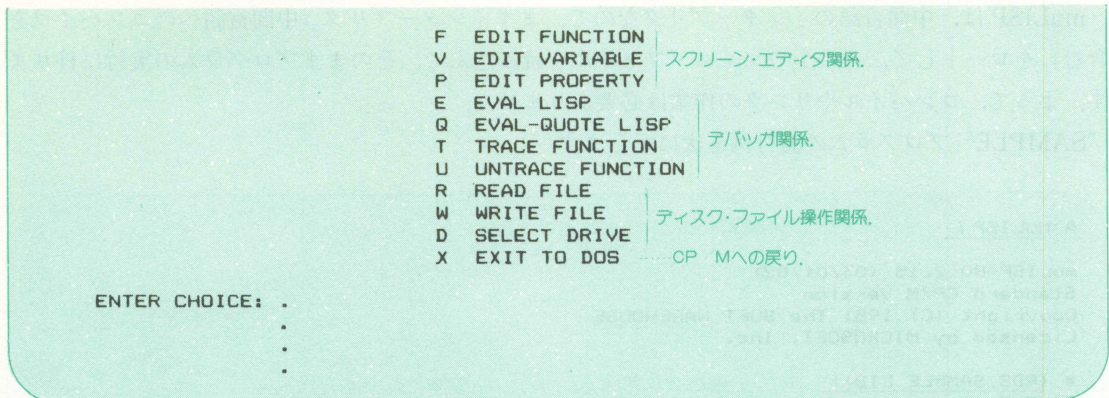



Figure-5.9.4 スクリーン・エディタ/デバッガの muSTAR を起動したところ。

muLISP の、その他の多くの機能の紹介は省略します。

5.10 ALGOL

5.10.1 ALGOLについて

ALGOL (ALGOrithmic Language) は、1958年に発表された科学技術計算用のプログラミング言語、IAL (International Algebraic Language) を基にして、内容の改訂と機能の拡張を行い、1960年にアルゴリズム記述言語 ALGOL 60として登場しました。これが現在の ALGOL の基となっています。

今日の進歩したプログラミング言語からみると、ALGOL はもはや歴史上のものとなってしまいましたが、プログラムの記述法やコンパイラのテクニックなど、以後の言語に与えた影響は大きく、ここでもあえて取り上げました。

5.10.2 Mark Moranville ALGOL-Mについて

ALGOL-M は、ALGOL-60を基に、マイクロコンピュータ上でのプログラミングに適するように作られた言語です。基本構成は ALGOL-60とほとんど同じであり、ALGOL-60で作られた従来のプログラムは、若干の書き換えで ALGOL-M 上で実行することが可能です。

ALGOL-M は、コンパイラ/インタープリタであり、ALGOL-M のソース・プログラムを中間コードにコンパイルし、それを実行時にインタープリタで実行します。実行時には、デバッグのための「トレース」の機能もあります。

ALGOL-M で使用できる変数は、 $-16383 \sim +16383$ の integer, 18 デジット以下の decimal, 255 文字以下の string の 3 つのタイプであり、デフォルトは decimal が 10 デジット, string が 10 文字となっています。

配列は 255 dimension までで、おのおのは $0 \sim 16383$ の値を持つことができます。

5.10.3 ALGOL-M による「SAMPLE」プログラムの作成

まず、ALGOL-M による「SAMPLE」プログラムのソース・ファイルを示します。

```
A>TYPE SAMPLE.ALG ..... ファイル名のエクステンションは必ず"ALG"であること.
begin ..... ALGOL のプログラムは"begin"で始まり"end"で終わる.

comment ALGOL-M sample program for "Application CP/M" ;
コメントは,"begin"のあと,または";"のあとの"comment~;"の間に自由に書く(少々めんどう).

integer num,numb,temp;          comment * variable declaration ;

write ("1+2+3+....n = x");      comment * print title ;
write ("input n ( n=1...180 )"); comment * print input message ;
read(num);                     comment * input n ;

                                comment * range check ;
while num > 180 do
begin
    write ("input n ( n=1...180 )");  ALGOL-M の整数は -16383 ~ +16383 の範囲しか扱えない.
    read(num);                       よって250ではオーバーするので180に変更してある.
end;

numb:= num;
temp:= num;

                                comment * compute ;
while num >= 1 do
begin
    numb:= num - 1;
    temp:= temp + num;
end;

write (" ");                    comment * print result ;
write ("1+2+3+....",numb," = ",temp);
write (" ");

end ..... このend以後は何を書いてもよい. 覚え書きなどがよく書かれる.

A>
```

Figure-5.10.1 ALGOL-M による「SAMPLE」プログラムのソース・ファイル.

次に ALGOL-M のパッケージの中から、「SAMPLE」プログラムの開発に必要な最低限のファイルを示しておきます。


```
A>B:STAT *.*

```

Recs	Bytes	Ext	Acc	
106	14k	1	R/W	A:ALGOLM.COM 中間言語へのコンパイラ.
112	14k	1	R/W	A:RUNALG.COM ランタイム・インタプリタ.
6	1k	1	R/W	A:SAMPLE.ALG "SAMPLE"プログラムのソース・ファイル.

```
Bytes Remaining On A: 212k
A>
```

Figure-5.10.2 ALGOL-M による "SAMPLE" プログラムの開発に必要な最低限の各ファイル.

ALGOL-M はインタプリタです. ソース・プログラムを "ALGOLM.COM" によって中間言語にコンパイルし, それをインタプリタの "RUNALG.COM" によって実行します.

では, 中間言語へのコンパイラの実行例を示します.

```
A>DIR SAMPLE.* / ..... 最初の"SAMPLE"ファイルの確認.
A: SAMPLE ALG ..... ソース・ファイルのみ存在.

A>ALGOLM SAMPLE / ..... コンパイラの実行.
ALGOL-M COMPILER VERS 1.1
0 ERROR(S) DETECTED
コンパイル成功.エラーなし.
A>DIR SAMPLE.* / ..... 生成されたファイルの確認.
A: SAMPLE ALG : SAMPLE AIN
   ソース・ファイル      生成された中間言語
                           のファイル
A>
```

Figure-5.10.3 ALGOL-M による中間言語へのコンパイル例.

コンパイルは成功して, 中間言語のファイル "SAMPLE.AIN" が生成されています.

これが ALGOL-M では, 最終的なファイルとなります.

では "SAMPLE" プログラムを実行してみましょう.

ALGOL-M は整数の範囲が-16383~+16383しか扱えないので, 入力180以上はエラーとなるように変更してあります.

```
A>RUNALG SAMPLE / ..... 中間言語のインタプリタによる"SAMPLE"プログラムの実行.
ALGOL-M INTERPRETER-VERS 1.0
```


$1+2+3+\dots n = x$
 input n (n=1...180)
 -> 300 / ALGOL-Mでは、コンソール入力には必ず復帰・改行が行われ、“/”がALGOL-Mのシステムにより出力される。
 input n (n=1...180)180以上の入力値エラー。
 -> 100 /今回は正しい値を入力。
 $1+2+3+\dots 100 = 5050$ 入力値と答えが出力された。
 A>

Figure-5.10.4 ALGOL-M により作成された“SAMPLE”プログラムの実行。

参考までに、リストアウト+トレース・モードのオプション・スイッチを付けて、コンパイラを実行した場合のコンソール出力を次に示しておきます。

```

A>ALGOLM SAMPLE $AE /
ALGOL-M COMPILER VERS 1.1
 1 0 begin
 2 1
 3 1 comment ALGOL-M sample program for "Application CP/M" ;
 4 1
 5 1
 6 1 integer num,numb,temp; comment # variable declaration ;
 7 1
 8 1 write ("1+2+3+....n = x"); comment # print title ;
 9 1 write ("input n ( n=1...180 )"); comment # print input message ;
10 1 read(num); comment # input n ;
11 1
12 1 comment # range check ;
13 1 while num > 180 do
14 1 begin
15 2 write ("input n ( n=1...180 )");
16 2 read(num);
17 1 end;
18 1
19 1 numb:= num;
20 1 temp:= num;
21 1
22 1 comment # compute ;
23 1 while num >= 1 do
24 1 begin
25 2 numb:= num - 1;
26 2 temp:= temp + num;
27 1 end;
28 1
29 1 write (" "); comment # print result ;
30 1 write ("1+2+3+....",numb," = ",temp);
31 1 write (" ");
32 1
33 1 end
34 0
35 0
36 0 EOF
0 ERROR(S) DETECTED
  
```

フロック・レベル
 ラインNo.

Figure-5.10.5 スイッチ“\$AE”（リストアウト+トレース）を付けてのコンパイラの実行。

ALGOL-M の、その他の多くの機能の紹介は省略します。

5.11 APL

5.11.1 APLについて

APL (A Programming Language) は、1956年から開発が始まり、1960年に対話形式のプログラミング言語として実用化されました。当初は IBM の内部で使用されており、一般には、1968年に IBM システム/360の TSS で使用され始めました。

APL は基本的には、0 (スカラー) 次元以上任意の配列データを対象に演算を行うもので、すべての操作は、関数の集まりで表現され、その文法は極めて単純であり、IF とか DO、各種宣言文などは一切ありません。

二項までの引数を持つ関数と、その関数の引数となるスカラー、もしくは1次元以上の任意の配列のみで構成され、その処理系はインタプリタの形式をとっています。よって、BASIC 言語のように、コンピュータと会話しながらプログラミングを行うことができ、APL の知識のない人でも、直接キーボードに向い、マニュアルに従ってキーインして行けば、即その答えが返ってくるので、効果的に学習することができます。

APL は、データ解析や数値解析の分野、ベクトルや配列演算を多く行わなければならないグラフィックスの分野などに、強力な言語として利用されています。

APL は、特種な文字や記号を持つキーボードを使いますので、そのモデル (IBM 2741 キーボード) を次に示しておきます。



Figure-5.11.1 APLの正式なキーボード

5.11.2 SOFTRONICS 社 APL\80について

SOFTRONICS の APL インタープリタは、フル APL のほとんどの機能を持っており、CP/M のディスク I/O のためのシステム・ファンクションと変数、システム・コマンドなどを備えています。

CP/M で APL を使うと言っても、あの特殊文字はどうするのかと疑問を持たれると思いますが、SOFTRONICS APL は、ターミナルの種類によって、“ASCII モード” と “APL ターミナル・モード” があり、APL の特殊文字を持たない普通の CP/M マシンでは、“ASCII モード” で実行することができます。

“ASCII モード” と言うのは、例えば、

ディメンションの	ρ ...	\$RO
逆行列の	$\begin{smallmatrix} \square & \square \\ \square & \square \end{smallmatrix}$...	\$XD
ローテートの	ϕ ...	\$RT

と言った具合に、特殊文字を \$ に続くアスキー標準文字の 2 字で代用します。入出力とも、これにより表現します。

“APL ターミナル・モード” と言うのは、IBM の 2741、2740、3767 などの APL 用のターミナルを接続できる場合のモードであり、正式の APL 文字による操作が行えます。CP/M マシンでは、キーボードに特殊文字を書いたステッカを貼り、スクリーンの表示にキャラクタをユーザーが任意に作り出すことができる VRAM などを使えばこのモードでの操作が可能です。APL のおもしろさが分かってくると、是非そうしたくなるでしょう（米国では、この APL 記号を表示できる S-100 バスの V-RAM ボードが市販されている）。

5.11.3 APL\80 による「SAMPLE」プログラムの作成

この APL\80 のおもしろさには、筆者もいささか興味を持ちましたので、本項は少々雰囲気を変えて解説したいと思います。

“APL\80” のバックスラッシュは、IBM の APL が、“APL\360” というように名付けられているので、それに従ったものと思われます。

実は、筆者は実際に APL を自分の手でいろいろ動かしてみるのには、この APL\80 が最初です。CP/M 上で走る APL があるということは以前から知っていましたが、実際に自分の S-100BUS マシンや、PC-8801 の CP/M で走らせてみると、この APL という言語は、実に興奮させるものがあります。

インタープリタ形式なので、普通のパーソナル・コンピュータの ROM に組み込まれている BASIC インタープリタと同様に、コンピュータと会話しながら非常に気軽にいろいろと試みることができ、楽しく学ぶことができます。

そんな APL の雰囲気を知って頂くために、キーボードに向って APL を起動し、ちょっと何かを実行してみましょう。

A>APLAPLを起動

□ユーザーのキー入力部分、キャリッジ・リタンを伴う。

SOFTRONICS APL*80
VERSION 2.3C

COPYRIGHT 1979 BY ERIK MUELLER

CLEAR WSワーキング・スペースは空というメッセージが出て、コマンドの入力待ちとなる。

25 **5+20**5+20

25 **5%20**5÷20

.25

I 5

.....Iに5を定義

J 20

.....Jに20を定義

J-I

.....J-I

15

I&J

.....I×J

100

A 3 3\$RO 1 2 3 4 5 6 7 8 9

.....変数Aに配列123...9を3×3の行列に変換して定義する。

A

.....そのタイプアウト。

1

2

3

4

5

6

行列がタイプアウトされる。

7

8

9

B 3 3\$RO 10 20 30 40 50 60 70 80 90

.....同じく変数Bに行列を定義。

B

.....そのタイプアウト。

10

20

30

40

50

60

70

80

90

B-A

.....行列の差をタイプアウト。

9

18

27

36

45

54

63

72

81

ATP \$TF A

.....行列Aを転置したものを変数ATPとして定義。

ATP

.....そのタイプアウト。

1

4

7

2

5

8

行列Aの転置行列。

3

6

9

AREV \$RT A

.....行列Aを反転したものを変数AREVとして定義

AREV

.....そのタイプアウト。

3

2

1

6

5

4

行列Aの反転行列。

9

8

7

***A**

.....行列Aを指数でタイプアウト(e^A)。

2.71828 7.38905 20.0855

54.5981 148.413 403.428

1096.63 2980.95 8103.06

ALOG \$LG A

.....行列Aの自然対数をとって、変数ALOGとして定義。

ALOG

.....そのタイプアウト。

0 .693147 1.09861

1.38629 1.60944 1.79176

ここで使用する代用記号の、正規のAPL記号を次に示します。

←←
%÷
x×
\$ROρ
\$TPτ
\$TRφ
\$LG⊗

1.94591 2.07944 2.19722

[↑] VARS 現在定義されている変数のすべてをタイプアウト。
I J A B ATP AREV ALOG
[↓] OFF CP/Mに戻る。
A>

Figure-5.11.2 APL の操作は大変簡単です。

このように実にシンプルな言語です。

さて、「SAMPLE」プログラムを、この APL\80で作成する訳ですが、そのソース・プログラムの作成は、本章で取り上げた他の言語のように、任意のエディタを使って、言語システムとは独立して別に作成することができません。ソース・プログラムは、APL システムに入って、その中でのみ作成されます。

インタープリタなので、作成中のプログラムの一部分を実行して、デバッグを行うことも可能です。実行中にバグがあれば、エラーのあるラインの内容を表示して、実行をストップします。

では、「APL\80の起動→「SAMPLE」プログラムの入力→実行→デバッグ実行→完成したプログラムのディスクへのセーブ」の手順を次に示します。

A>APL APL.COMを起動。 ☐部はユーザーのキーイン部を示し、キャリッジリタンを伴う。

SOFTRONICS APL¥80
VERSION 2.3C

↑ 本来は/バックスラッシュ"/である。日本のプリンタでは実にサマにならない(80円のAPL?)。

COPYRIGHT 1979 BY ERIK MUELLER

→ APLでは、システムの応答はこの位置から表示される。

CLEAR **[WS]** ワーク・スペースには何も無いことを示す。

[none] **[↑] FNS** 現時点で定義されている関数をすべてタイプアウトせよ。結果は空白の応答で何も定義されていない。
 → APLでは、ユーザーのキーインするものはこの位置から表示される。

[1] S **[↑] \$DL SAMPLE; N; NUMB; TEMP** "SAMPLE"プログラムを関数SAMPLEとして定義モードに入る。

[2] A **[↑] ' ';** 改行。その引数はない。

[3] M **[↑] '1+2+3+...n = x'** タイトルの出力。

[4] P **[↑] READ: 'input n (n = 1...250) --->'** 入力メッセージの出力とラベル"READ"の宣言。

[5] L **[↑] \$GO (250<N NUMB TEMP \$EV \$OP)/, READ** 数字を入力してN NUMB, TEMPに代入。値が250より大の

[6] E **[↑] COMPUTE: TEMP TEMP+NUMB NUMB-1** NUMB←NUMB-1, TEMP←TEMP+NUMB。時はREADへ帰る。

[7] R **[↑] \$GO (0\$NE NUMB) /, COMPUTE** NUMB≠0ならCOMPUTEへ。

[8] T **[↑] ' ';** 改行。

[9] U **[↑] \$QD '1+2+3+... ', (\$FM N), / = ', (\$FM TEMP)** 入力値と答への出力。

[10] V **[↑] ' ';** 改行。

[11] W **[↑] \$DL** 定義モードを終了せよ。

[12] X **[↑] \$FNS** 前出。

SAMPLE

[SAMPLE] 関数SAMPLE(引数はない)の実行。つまり"SAMPLE"プログラムの実行。

1+2+3+...n = x

input n (n = 1...250) --->

[300] 251以上の入力値エラー。APLでは入力時は必ず改行が行われる。

input n (n = 1...250) --->


```

[251] -----入力値エラー.
input n (n = 1...250) ---->
[250] -----正しい値を入力.

SYNTAX ERROR -----プログラムにバグあり!
SAMPLE[8] $QD _'1+2+3+....', ($FM N), /=' , ($FM TEMP) -----エラーのあるラインが表示される.

[10] [DL SAMPLE] -----編集モードに入れ.
[8] [B$QD] -----ライン[8]を表示して、再入力モードに入れ。ただし1行全部をタイプし直す必要あり.
[8] $QD _'1+2+3+....', ($FM N), /=' , ($FM TEMP)
[8] $QD _'1+2+3+....', ($FM N), ' = ' , ($FM TEMP) -----めんどくさくても1行全部をタイプし直す.
[9] [DL] -----編集モードを終われ.
[SAMPLE] -----もう一度実行.

1+2+3+....n = x
input n (n = 1...250) ---->
[250] -----250を入力.

1+2+3+....250 = 31375 -----入力値と答えが正しく出力された(そのままCP/Mに戻ることはできない).

[SAMPLE] -----もう1度実行.

1+2+3+....n = x
input n (n = 1...250) ---->
[100]

1+2+3+....100 = 5050 -----正しい!

[SAVE SAMPLE] -----"SAMPLE"プログラムをディスクにセーブする。正しく表現すると、現在のWS(ワーキング・ス
SAMPLE SAVED -----ベース)の全部をセーブする。これは関数SAMPLEのみをセーブするのではなく、この時点で定義
[OFF] -----CP/Mに戻る。済みのすべての関数や変数(この例では"SAMPLE"のみしか存在しないが)をセーブするものであ
り、後でこれをロードすると、セーブした時点と同じ状態を再現できる。

```

Figure-5.11.3 APL\80による "SAMPLE" プログラムの作成の全過程。

以上の手順で "SAMPLE" プログラムはでき上り、実際に使用テストも行いました。ディスク上には、そのプログラム・ファイル (アスキー・ファイルではなく、中間コード) "SAMPLE. APL" が生成されています (ここで使用した APL 記号の代用表示については後述)。

現在のディスク上には、APL\80のパッケージの中から、必要なファイルだけと、でき上った "SAMPLE" プログラムがセーブされていますので、それらを STAT コマンドでタイプアウトしてみましょう。

```

A>B:STAT *.*!

Recs  Bytes  Ext  Acc
 284   36k    3  R/W  A: APL.COM -----APLインタープリタ.
  23    3k    1  R/W  A: ETCFNS.APL -----各種関数のライブラリ, "SAMPLE"プログラムには必要なし.
   6    1k    1  R/W  A: SAMPLE.APL -----でき上った"SAMPLE"プログラム.
Bytes Remaining On A: 201k

A>

```

Figure-5.11.4 ディスク上に現在セーブされているファイル。


```

$DL HISTO [$QD] $DL .....ヒストグラム(HISTO)のプログラムをタイプアウトせよ。
$DL Z_SYM HISTO V
[1] $
[2] Z_(' ',SYM)[1+($RT $IO $MX /V)$NL . $LE V] .....たった1行で上記の機能が書ける, APLはスゴイ!
$DL
$DL SORT [$QD] $DL .....ソート(SORT)の内容をタイプアウトせよ。
$DL ORD_SORT UNS; WHICH
[1] ORD_$IO 0
[2] LB:$GO 0&$IO 0=$RO UNS
[3] WHICH_UN$=$MN /UNS
[4] ORD_ORD, WHICH/UNS
[5] UNS_($TL WHICH)/UNS
[6] $GO LB
$DL
[PASCAL 20] .....大きさが20のパスカルの三角形をタイプアウトせよ。
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
1 16 120 560 1820 4368 8008 11440 12870 11440 8008 4368 1820 560 120 16 1
1 17 136 680 2380 6188 12376 19448 24310 24310 19448 12376 6188 2380 680 136 17 1
1 18 153 816 3060 8568 18564 31824 43758 48620 43758 31824 18564 8568 3060 816 153 18 1
1 19 171 969 3876 11628 27132 50388 75582 92378 92378 75582 50388 27132 11628 3876 969 171 19 1

$DL PASCAL [$QD] $DL .....そのプログラムをタイプアウトせよ。
$DL PASCAL N;P
[1] P_1
[2] PRINT:P
[3] $GO PRINT&N$GE $RO P_(0,P)+P,0
$DL
[OFF] .....遊びはやめてCP/Mに戻れ。

```

A> (注: ORTや, プリンタへの出力は, 1行の字数が多いと, 次の行に続いて表示されます.)

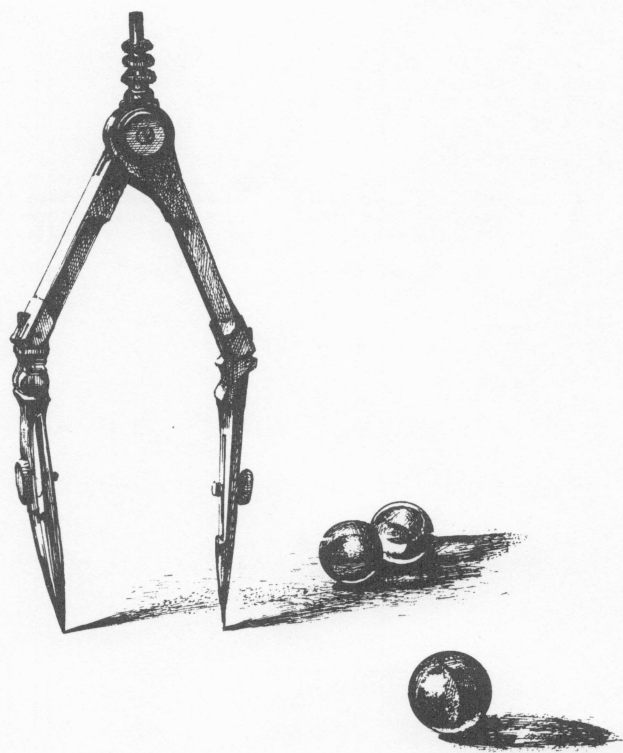
Figure-5.11.5 APL\80でちょっと遊びを。

Figure-5.11.3や Figure-5.11.5で使われた APL 記号の代用 ASCII 表示は, 実際は次の記号に当たります。

→...\$GO	□...\$QD
←..._ (下線)	▽...\$DL
Φ...\$EV	≠...\$NE
⊖...\$FM	ρ...\$RO
⌈...\$QP	

Figure-5.11.6 "SAMPLE"プログラムの作成に使われたAPL記号の代用ASCII表記

6章 CP/Mのアプリケーションいろいろ



6.1 簡易言語(プログラムレス言語)の使用例

VISICALC (Visi Corp 社) に端を発した、“簡易言語”と呼ばれている汎用ビジネス・ソフトウェアは、その質・内容・種類とも大きな進展を見せており、現時点では、“Multiplan” (マイクロソフト社)、“SuperCalc” (ソーシム社) などが、質・内容ともに代表的なものになっています。

筆者は、米国製の、この2種類のソフトを実際に使ってみて、日米間のソフトウェア・ギャップ、正にここにあり！という強い印象を受け、このことを「実習 CP/M」の冒頭にも書きました。

日本でも、この種のソフトは雨後の竹の子のようにたくさんの種類が発売され、中には、大々的に宣伝されているものもあります。しかし、そのほとんどは、特定のパーソナル・コンピュータ専用のもので、プログラムも、その機種に組み込まれている BASIC のインタープリタで、ただズラズラと書かれているものばかりでスピードも遅く(マシン語をプログラムの一部に使っているものもあるが)、内容、使い易さなど、先の2つの米国製ソフトに比べあまりにも幼い、というのが実感です。

CP/M 上で走る優秀なソフトがあることを知らず、これらの貧弱なソフトが、パソコンの能力の限界であるなどと誤解する人もいるのではないかと心配です。

それでは、この種のソフトの中で、米国でベストセラーの1つと言われている、ソーシム社の SuperCalc を取り上げて、その使用例を紹介しましょう。しかし、そのすべてを紹介する訳には行きません。例によって、すべてを紹介するには、SuperCalc 用の本を一冊書かなくてはならないでしょう。よって、ここで例題としたものは、機能の一部、使い方の一例であるに過ぎません。

6.1.1 SuperCalcの概念

VISICALC, Multiplan, SuperCalc などの汎用ビジネス・ソフトの概念は、どれもほぼ同じであり、“ワークシート”と呼ばれる“電子の紙”(コンピュータのメモリのデータ・エリアに当たる)を SuperCalc では、横63×縦254の計16,002個の“セル”に区切り、その“セル”に、項目や数値などを書き込み、“電子の紙”を“表”として用い、縦・横の計算(各種関数も含む)などを行わせるものです。1つの表の中でのデータのブロック移動や、他の場所へのコピー、他の表からのデータの取り込みなども自由に行えます。

Figure-6.1.2の表が、その“電子の紙”の実際であり、SuperCalc を起動した状態では、表全体の左上に当る“ディスプレイ・ウィンドウ”が、スクリーンに表示されています。

表の上の位置の呼び方は、横方向に“カラム (Column) A, B, C……” 縦方向に“ロー (Row) 1, 2, 3……”と表現します。それぞれの“セル”はカラムとローの交点に当たり、この図に示されているセルは、“E10”のセルということになります。

この図の、セル“A1”の“< >”記号（コンピュータによっては、この部分の表示が反転したり、色が付いていたりする）は、“アクティブ・セル”を示す記号であり、ワークシートへのデータの入力や、書き替えなどは、この“アクティブ・セル”を通して行われます。

6.1.2 SuperCalcの使用例

では、SuperCalc を起動して、実際に簡単な表を作ってみましょう。NEC の PC-8801上で、NEC の CP/M を使って実行してみます。

SuperCalc の起動は、

A>SC J

で行われます。その、オープニング・メッセージを次に示します。

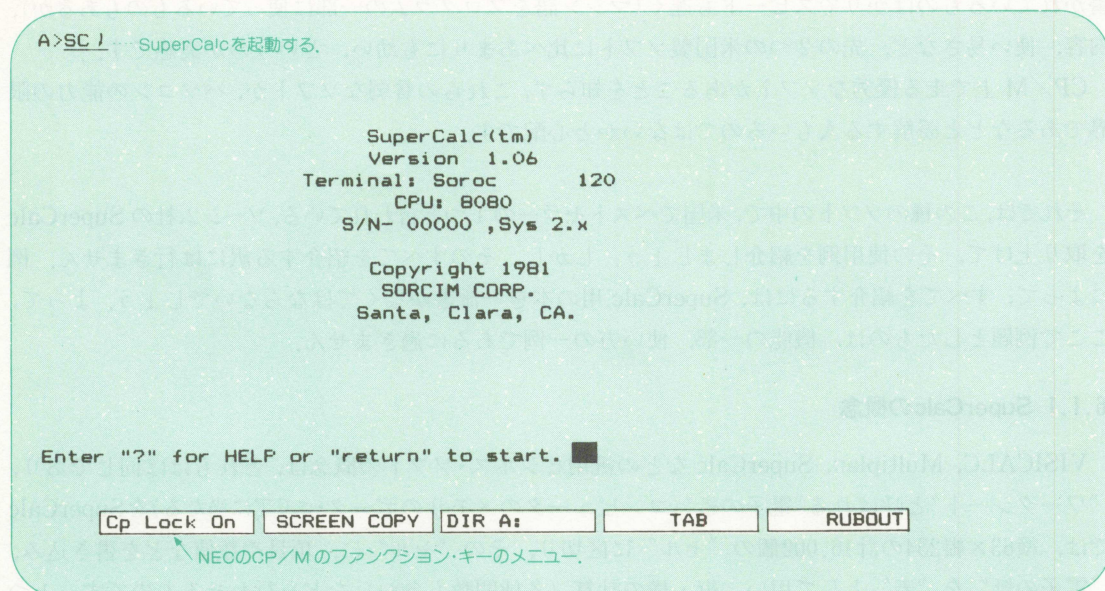


Figure-6.1.1 SuperCalc の起動と、そのオープニング・メッセージ。

さっそく“RETURN”をキーインして、プログラムをスタートさせます。

プログラムがスタートすると、スクリーンには、Figure-6.1.2の左上方に示されている“ディスプレイ・ウィンドウ”の部分が表示されます。この部分は、図でも分かるように、ワークシート全体の、ほんの一部分であり、ユーザーは、大きなワークシートの任意の場所を、このディスプレイ・ウィンドウを上下左右に移動することにより、見ることができる訳です。

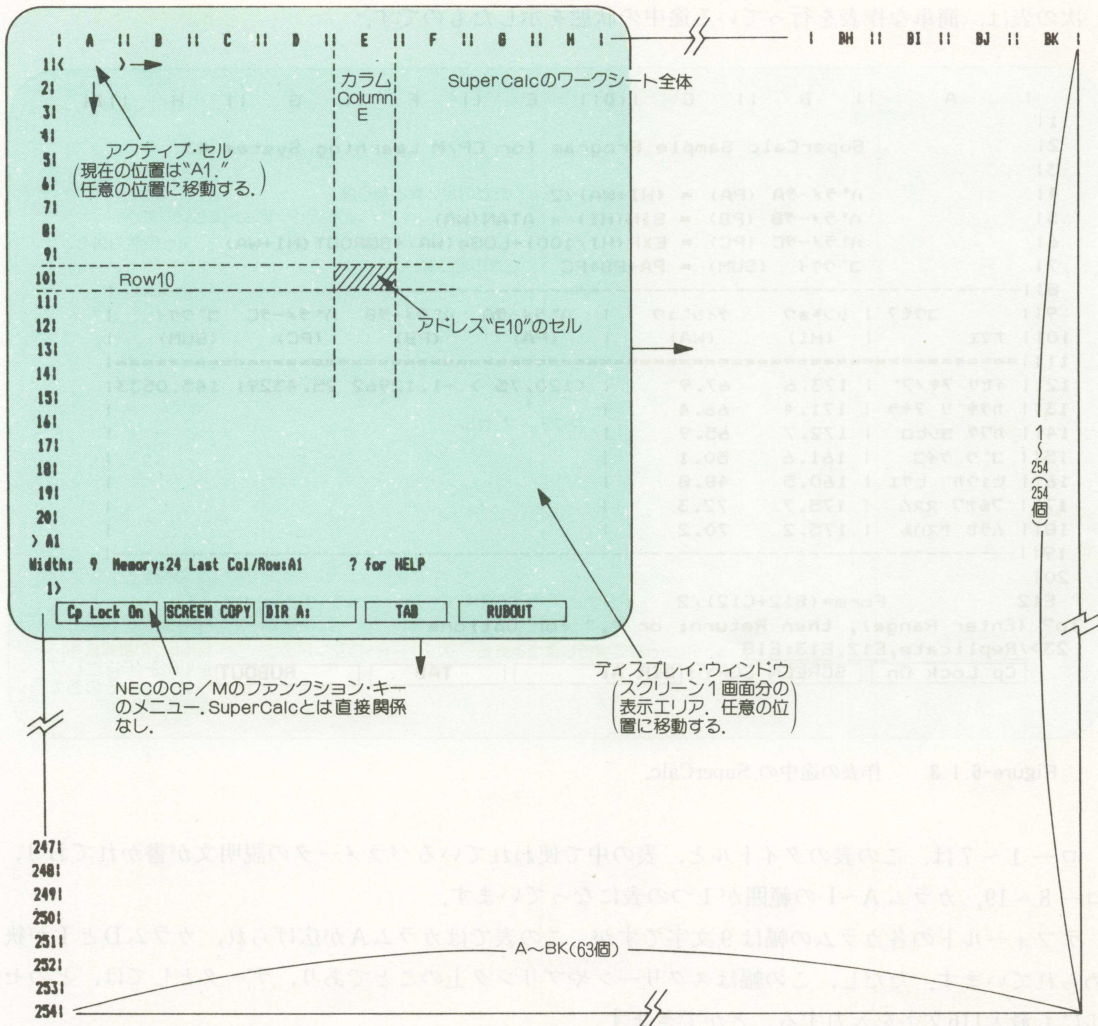


Figure-6.1.2 SuperCalcのワークシートとディスプレイ・ウィンドウ

次の表は、簡単な作表を行っている途中の状態を示したものです。

	A	B	C	D	E	F	G	H	I
1	SuperCalc Sample Program for CP/M Learning System #3								
4	パラメータ (PA) = (HI+WA) / 2 ただの足し算と割り算.								
5	パラメータ (PB) = SIN(HI) x ATAN(WA) サインとアーク・タンジェント(単位はラジアン)								
6	パラメータ (PC) = EXP(HI/100)+LOGe(WA)+SQROOT(HI+WA) eと自然対数と√								
7	コウケイ (SUM) = PA+PB+PC ただの足し算.								
9	コウモク	シンショウ	タイシユウ	パラメータ	パラメータ	パラメータ	コウケイ		
10	ナマエ	(HI)	(WA)	(PA)	(PB)	(PC)	(SUM)		
12	イセリ アキノブ	173.6	67.9	<120.75 >	-1.12962	25.43291	145.0533		
13	カタギリ アキラ	171.4	66.4	↑					
14	カワタ ヨシヒロ	172.7	65.9	アクティブ・セル					
15	コウケイ	161.6	50.1						
16	ヒョウカ ヒサエ	160.5	48.8						
17	フルカワ スズム	175.7	72.3						
18	ムラセ ヤスハル	175.2	70.2						
20	Form=(B12+C12)/2 アクティブ・セルの計算式(フォーミュラ)が表示されている.								
To? (Enter Range), then Return; or ", " for Options									
23>/Replicate,E12,E13:E18 次に実行しようとするレプリケート・コマンドのライン. セルE12の計算式を, セルE13~E18にレプリケートし, その答えを表示する.									
Cp Lock On		SCREEN COPY		DIR A:		TAB		RUBOUT	

Figure-6.1.3 作表の途中の SuperCalc.

ロー 1~7 は、この表のタイトルと、表の中で使われているパラメータの説明文が書かれており、ロー 8~19, カラム A~I の範囲が 1 つの表になっています。

デフォルトの各カラムの幅は 9 文字ですが、この表ではカラム A が広げられ、カラム D と I が狭められています。ただし、この幅はスクリーンやプリンタ上のことであり、データとしては、どのセルにも最大 116 文字を入力することができます。

表には、実在の人物の氏名とてたらめの身長・体重の値が記入されており、その右側には、意味のないパラメータ A, B, C と、その合計の項目があります。各パラメータが、どのような計算式により求められるかは、ロー 4~7 に書かれています。それらの計算式 (フォーミュラ) は、すでに「イセリ アキノブ」の行のみに書き込まれており、その計算結果が表示されています。パラメータ A (PA) の実際のフォーミュラを、表のすぐ下に見ることができます。

「カタギリ アキラ」以後は、データのみが記入されているだけで、フォーミュラは記入されておらず、その計算結果は白紙のままとなっています。

では、パラメータ A のカラムを全部埋めてみましょう。

前頁の表の下部には、そのコマンド・ラインがすでに書かれています。

／Replicate, ……複写する。

E12, E13 : E18……セルE12のフォーミュラをセル E13～E18にレプリケートする。

という意味のコマンド・ラインです。

どうなるか、リタンをキーインして実行してみましょう。

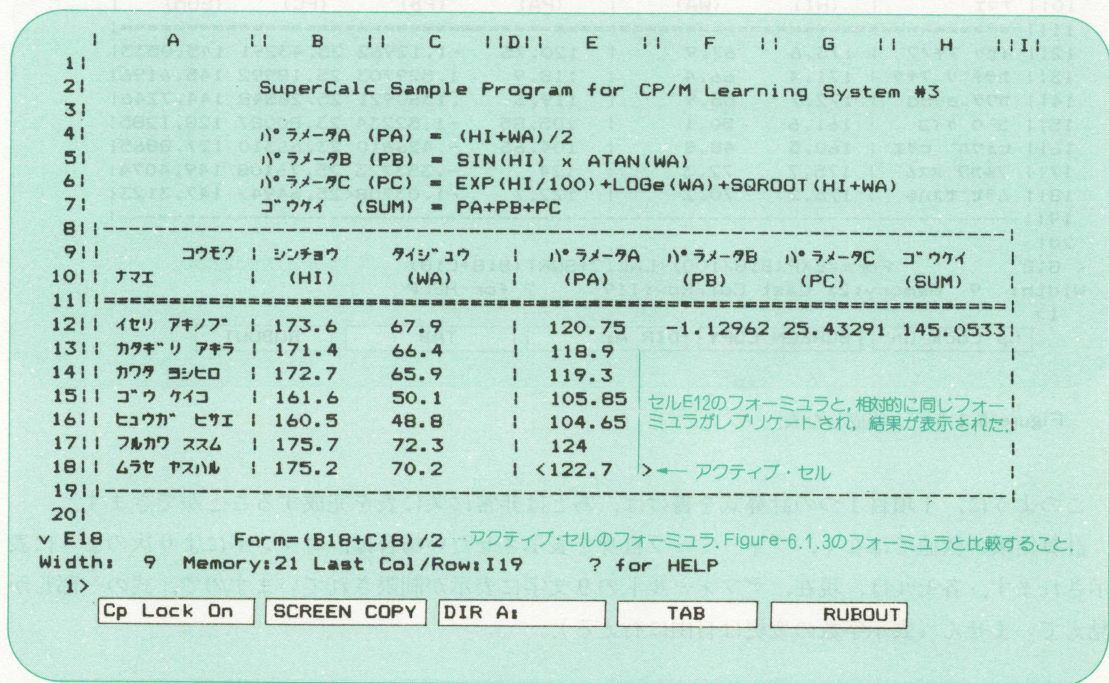


Figure-6.1.4 レプリケート・コマンドの実行。

このように、セル E13～E18がすべて計算され、結果が表示されました。つまり、セル E12のフォーミュラが、これらのセルに相対アドレスでレプリケートされたのです。セル E18のフォーミュラが、スクリーンの下部に表示されていますので、Figure-6.1.3のものと比較して下さい。

同様に、「イセリ アキノブ」の行の、すでに書き込まれている各パラメータのフォーミュラをレプリケートして、完成させた表を次に示します。


```

1|      A      | B      | C      | D      | E      | F      | G      | H      |
2|      SuperCalc Sample Program for CP/M Learning System #3
3|
4|      ハ*ラメータA (PA) = (HI+WA)/2
5|      ハ*ラメータB (PB) = SIN(HI) x ATAN(WA)
6|      ハ*ラメータC (PC) = EXP(HI/100)+LOGe(WA)+SQROOT(HI+WA)
7|      コ*ウケイ (SUM) = PA+PB+PC
8|-----|
9|      コウモク | シンチョウ | タイシ`ユウ | ハ*ラメータA | ハ*ラメータB | ハ*ラメータC | コ*ウケイ |
10| ナマエ      | (HI)       | (WA)       | (PA)       | (PB)       | (PC)       | (SUM)     |
11|-----|
12| イセリ アキノフ | 173.6     | 67.9       | 120.75     | -1.12962   | 25.43291   | 145.0533 |
13| カタキ`リ アキラ | 171.4     | 66.4       | 118.9       | 1.529703   | 25.18992   | 145.6196 |
14| カワタ ヨシヒロ | 172.7     | 65.9       | 119.3       | .1360921   | 25.28848   | 144.7246 |
15| コ`ウ ケイコ | 161.6     | 50.1       | 105.85     | -1.52234   | 23.80087   | 128.1285 |
16| ヒュウカ` ヒサエ | 160.5     | 48.8       | 104.65     | -.426610   | 23.66310   | 127.8865 |
17| フルカワ スズム | 175.7     | 72.3       | 124         | -.353723   | 25.76108   | 149.4074 |
18| ムラセ ヤスハル | 175.2     | 70.2       | 122.7      | -1.03708<25.6494> | 147.3123 |
19|-----|
20|
< G18      Form=EXP(B18/100)+LNC12+SQRT(B18+C18) .....パラメータCのセルG18のフォーミュラ.
Width: 9  Memory:21 Last Col/Row:I19      ? for HELP
1>

```

Cp Lock On SCREEN COPY DIR A: TAB RUBOUT

Figure-6.1.5 完成した表.

このように、1項目1つの計算式を書けば、あとは非常に楽に表を完成することができます。

計算結果の数値ではなく、フォーミュラ自身を表示させたい場合は、コマンドにより次のように表示されます。各セルは、現在、デフォルトの9文字に表示が制限されていますので、式の一部しか見えていません（表示字数の変更は自由に行える）。

```

1|      A      | B      | C      | D      | E      | F      | G      | H      |
2|      SuperCalc Sample Program for CP/M Learning System #3
3|
4|      ハ*ラメータA (PA) = (HI+WA)/2
5|      ハ*ラメータB (PB) = SIN(HI) x ATAN(WA)
6|      ハ*ラメータC (PC) = EXP(HI/100)+LOGe(WA)+SQROOT(HI+WA)
7|      コ*ウケイ (SUM) = PA+PB+PC
8|-----|
9|      コウモク | シンチョウ | タイシ`ユウ | ハ*ラメータA | ハ*ラメータB | ハ*ラメータC | コ*ウケイ |
10| ナマエ      | (HI)       | (WA)       | (PA)       | (PB)       | (PC)       | (SUM)     |
11|-----|
12| イセリ アキノフ | 173.6     | 67.9       | (B12+C12 SIN(B12) EXP(B12/ SUM(E12+
13| カタキ`リ アキラ | 171.4     | 66.4       | (B13+C13 SIN(B13) EXP(B13/ SUM(E13+
14| カワタ ヨシヒロ | 172.7     | 65.9       | (B14+C14 SIN(B14) EXP(B14/ SUM(E14+
15| コ`ウ ケイコ | 161.6     | 50.1       | (B15+C15 SIN(B15) EXP(B15/ SUM(E15+

```

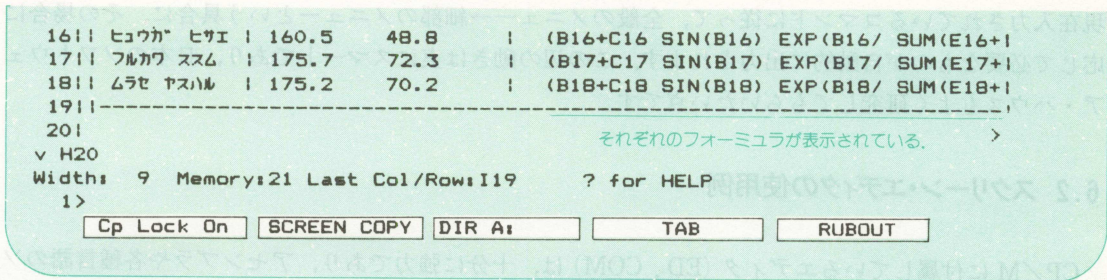



Figure-6.1.6 数値の代わりに、そのフォーミュラを表示させたもの。

表がすべて完成し、プリンタへタイプアウトした結果を示します。ロー、カラムの表示は、付けたり取ったり自由です。

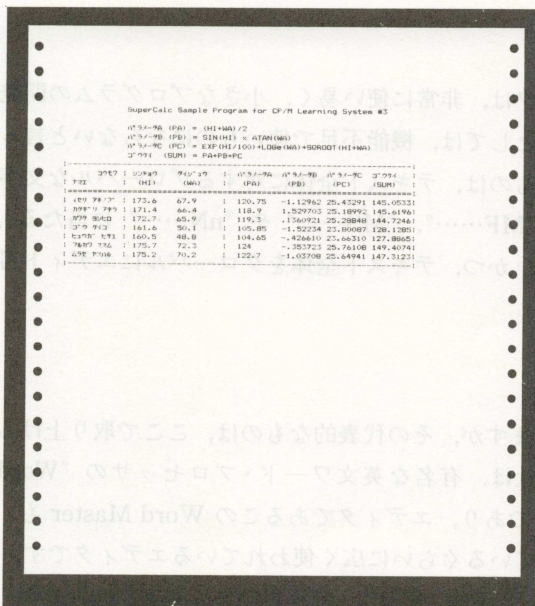


Figure-6.1.7 タイプアウトされた完成した表。

以上紹介したのは、SuperCalc の使い方のごく基本的な簡単な例にすぎません。まだまだ目を見張るような実に多くの機能を有していることを、お断りしておきます。

さらに SuperCalc には、メニュー・システムが完備されていて、使い方や、それぞれのコマンドが分からない場合には、"? " 記号をキーインすることにより、その場で必要なコマンド・メニューや、コメントをいつでも表示することができます。このメニュー・システムはツリー構造になっていて、

現在入力されているコマンドに従って、全般のメニュー→細部のメニューという具合に、その場合に
応じて必要なものが自動的に出力されます。この辺の動きは実にスマートであり、日本のソフトウェ
ア・ハウスもよく研究してもらいたい点です。

6.2 スクリーン・エディタの使用例

CP/M に付属しているエディタ (ED.COM) は、十分に強力であり、アセンブラや各種言語のソ
ース・ファイルの作成に広く使われています。ED は強力ではありますが、それを自由自在に使いこな
すには、かなりの熟練を必要とします。筆者も、自在に使えるようになるまでは、恐らく 1 年ぐら
いはかかったような気がします。

自在に使いこなせるようになると、ED の強力さは、一般のパーソナル・コンピュータに組み込みの
BASIC が持っているエディタ (そのほとんどがスクリーン・エディタ) とは格段の差があることがよ
く分かるようになります。

一般の BASIC が持っているスクリーン・エディタは、非常に使い易く、小さなプログラムの開発
には大変便利ですが、大きなプログラムのエディタとしては、機能不足で使いものにならないと言っ
ても過言ではありません。致命的な不足機能の主なものは、テキスト全体に対するグローバルな文字
列サーチと文字列の置き換えです (ED のコマンドの "MF.....", "MS....." や "nN....." に当たる)。

そこで、使いやすいスクリーン・エディタであり、かつ、テキスト全体をグローバルにエディット可
能なエディタが望まれることになります。

6.2.1 Micro Pro社のWord Master

スクリーン・エディタは、数社から発売されていますが、その代表的なものは、ここで取り上げる
Micro Pro 社の Word Master です。Micro Pro 社は、有名な英文ワード・プロセッサの "Word
Star" で広く知られているソフトウェア・メーカーであり、エディタであるこの Word Master も、
CP/M 上のスクリーン・エディタの代名詞になっているぐらいに広く使われているエディタです。

Word Master は、誰でもすぐ使える便利なスクリーン・エディタとテキスト全体をグローバルにエ
ディットする機能を合わせ持っており、そのために、2つのエディット・モードを備えています。

- VIDEO モード
- COMMAND モード

の2つであり、VIDEO モードがスクリーン・エディタのモードであり、COMMAND モードが、CP/

M に付属の ED と、だいたいコマンド・コンパチブルなポインタ形式のエディタのモードで、グローバルなエディットが可能です。エディット作業は、この2つのモードを、場合に応じて切り替えながら進めて行く訳です。モード切り替えは簡単であり、VIDEO モードから COMMAND モードへは“ESC”キーの入力、その逆は“V J”のキー入力により瞬時に切り替わります。

6.2.2 Word Masterの実行例

実際に Word Master を起動して、CP/M に付属している DUMP プログラムのソース・ファイル“DUMP. ASM”をエディットしてみましょう。CP/M マシンには、PC-8801を使い、NEC の CP/M を走らせています。以下のリストは、その SCREEN COPY の機能によりタイプアウトしたものです。

Word Master を実行するには、2つのファイル、

WM. COM

WM. HLP (ヘルプ・メニューの表示をしなければ、これは必要ない)

が必要です。

Word Master がドライブA:、“DUMP. ASM”がドライブB: 上にあるとして、Word Master を起動します。その様子と、起動直後、ほんのわずかの間表示される製品メッセージを次に示します。

```
A>WM B:DUMP.ASM/
MicroPro WORDMASTER release 1.07A serial # WM2510SV
COPYRIGHT (C) 1978 MICROPRO INTERNATIONAL CORPORATION
```

ドライブB: 上にある、DUMPプログラムのソース・ファイルに対して、ワードマスタを起動。
起動が完了すると、ほんのちよつとの間、このメッセージが現れ、画面全体がクリアされ、次の画面となる。

Cp Lock On SCREEN COPY DIR A: TAB RUBOUT

Figure-6.2.1 Word Masterの起動とオープニング・メッセージ。

ほんの少しの間、上のメッセージが現れ、その後すぐに画面のオール・クリアが行われ、テキストである“DUMP. ASM”の最初の1ページが表示されます。

その状態を次に示します。これは自動的に行われ、EDのように、アペンドのコマンドを実行する必要はありません。以後も、アペンドやセーブは自動的に行われるので、EDのAコマンドやWコマンドに相当するものは全く必要ありません。

起動した直後のカーソルの位置。

```

;
; FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
; COPYRIGHT (C) 1975, 1976, 1977, 1978
; DIGITAL RESEARCH
; BOX 579, PACIFIC GROVE
; CALIFORNIA, 93950
;
;
; ORG      100H
BDOS      EQU      0005H    ; DOS ENTRY POINT
CONS      EQU      1        ; READ CONSOLE
TYPEF     EQU      2        ; TYPE FUNCTION
PRINTF    EQU      9        ; BUFFER PRINT ENTRY
BRKF      EQU      11       ; BREAK KEY FUNCTION (TRUE IF CHAR READY)
OPENF     EQU      15       ; FILE OPEN
READF     EQU      20       ; READ FUNCTION
;
FCB        EQU      5CH     ; FILE CONTROL BLOCK ADDRESS
BUFF      EQU      80H     ; INPUT DISK BUFFER ADDRESS
;
; NON GRAPHIC CHARACTERS
CR         EQU      0DH     ; CARRIAGE RETURN
LF         EQU      0AH     ; LINE FEED

```

画面全体がクリアされて、このようにエディットされるテキストの最初の1ページが表示される。カーソルは左上端にある。

Cp Lock On SCREEN COPY DIR A: TAB RUBOUT

Figure-6.2.2 WMが起動して、テキストの1ページ目が表示されたところ。

この後、2～3の簡単なエディットを実行してみますが、その前に、どのようなコマンドがあるのかを、ヘルプ・メニューで見ておきましょう。

Word Masterのヘルプ・メニューは、いつでもどんな時でもCtrl-Jをキーインすることにより表示させることができます。

Ctrl-J をキーインすると、まず次のコマンド・メニューが表示されます。

```

VIDEO MODE SUMMARY   (TYPE ^J FOR NEXT FRAME)

^O   INSERTION ON/OFF                    RUB   DELETE CHR LEFT
^S   CURSOR LEFT CHAR                   ^G   DELETE CHR RIGHT
^D   CURSOR RIGHT CHAR                   ^*   DELETE WORD LEFT
^A   CURSOR LEFT WORD                   ^T   DELETE WORD RIGHT
^F   CURSOR RIGHT WORD                   ^U   DELETE LINE LEFT
^Q   CURSOR RIGHT TAB                   ^K   DELETE LINE RIGHT
^E   CURSOR UP LINE                     ^Y   DELETE WHOLE LINE
^X   CURSOR DOWN LINE                   ^I   PUT TAB IN FILE
^^   CURSOR TOP/BOT (^HOME)            ^N   PUT CRLF IN FILE
^L   CURSOR RIGHT/LEFT                  ^@   DO NEXT CHR 4X

```


^W	FILE DOWN 1 LINE	^P	NEXT CHR IN FILE
^Z	FILE UP 1 LINE	^V	VIO CONTROL
^R	FILE DOWN SCREEN	ESC	EXIT VIDEO MODE
^C	FILE UP SCREEN	^J	DISPLAY THIS

コマンドが分からない場合は、いつ、どんな時でもCtrl-Jをキーインすることにより、このコマンド・メニューが表示される。
このメニューは、ビデオ・モード(スクリーン・エディタのモード)のページであり、さらにCtrl-Jをキーインすることにより、次ページのメニューが表示される。
Ctrl-J以外のものをキーインすると、元の状態のままのテキストに戻ることができる。

Cp Lock Offf SCREEN COPY DIR A: TAB RUBOUT

Figure-6.2.3 ヘルプ・メニューの最初のページ。VIDEO モードのコマンド・メニュー。

続いて同じく Ctrl-J をキーインすると、次のページのコマンド・メニューが表示されます。

COMMAND MODE SUMMARY (TYPE ^J FOR NEXT FRAME)

+ - MEANS + OR - ALLOWED HERE, + ASSUMED IF OMITTED
 @ MEANS CARRIAGE RETURN OR LINE FEED NECESSARY HERE
 \$ MEANS ESC OR ^Z OR CARRIAGE RETURN NECESSARY HERE
 n MEANS A NUMBER, 1 ASSUMED IF OMITTED, # = 65535

+ -nC	MOVE n CHARACTERS	+ -nD	DELETE n CHARACTERS
+ -nL	MOVE n LINES	+ -nK	KILL (DELETE) n LINES
+ -nT	TYPE n LINES	nZ	SLEEP n SECONDS
+ -nP	MOVE, TYPE n PAGES	+ -n@	MOVE n LINES, TYPE 1

nItext\$ INSERT text n TIMES
 I@ ENTER INSERT MODE (ESC OR ^Z EXITS MODE)
 A@, nAtext\$ (APPEND) DO 1L THEN JUST LIKE INSERT
 n<....> LOOP: REPEAT n TIMES (DEFAULT = 65535)

コマンド・メニューの2ページ目。コマンド・モード(ポインタ形式のエディタ・モード)のメニューの1ページ目。さらにCtrl-Jをキーインすることにより、あと2ページのコマンド・メニューを表示させることができる。

Cp Lock Offf SCREEN COPY DIR A: TAB RUBOUT

Figure-6.2.4 2ページ目。COMMAND モードのコマンド・メニュー。

では、2～3の簡単なエディット作業を示します。内容はリスト内の解説文を参照して下さい。また、Figure-6.2.3のコマンド・メニューも参照して下さい。

Cp Lock Off SCREEN COPY DIR A: TAB RUBOUT

Figure-6.2.6 スクリーン・エディット例.

Word Master のスクリーンには、メモリ上のエディット・バッファの一部が表示されており、スクリーン上の変更は、すなわち、エディット・バッファのテキストの変更である訳です。

次にVIDEO モードから、COMMAND モードに切り替えてみましょう。"ESC" キーの入力と同時にCOMMAND モードに切り替わります。

```

      ORG      100H
BDOS   EQU     0005H   ;DOS ENTRY POINT
CONS   EQU     1       ;READ CONSOLE
TYPEF   EQU     2       ;TYPE FUNCTION
PRINTF   EQU     9       ;BUFFER PRINT ENTRY
BRKF    EQU     11      ;BREAK
OPENF   EQU     15      ;FILE abcdefg
READF   EQU     20      ;READ CTION
;
FCB     EQU     5CH     ;FILE CON12345TROL BLOCK ADDRESS
BUFF    EQU     80H     ;INPUT DISK BUFFER ADDRESS
;
/*ESC*/キーの入力により、画面左下端に"*/"が表示され、コマンド・モードに切り替わる。"*/"はコマンド・
モードのプロンプト。現在はスクロールupされ、この位置に来た。
*10T/ .....CP/Mの"ED"と同様に、10ライン分をタイプアウトする。コマンド・モードでの実行。
;
      FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
;
      COPYRIGHT (C) 1975, 1976, 1977, 1978
;
      DIGITAL RESEARCH
;
      WordMaster sample run.
;
      Ctrl-N = Insert line.
;
      Line insert sample for "Application CP/M".
;
      BOX 579, PACIFIC GROVE
;
      CALIFORNIA, 93950
;
;
* → コマンド・モードのプロンプト。 → ビデオ・モードに戻るには、"VI"をキーインする。
Cp Lock Off SCREEN COPY DIR A: TAB RUBOUT

```

10ライン分が
タイプアウト
されている。

Figure-6.2.7 ESCキーの入力によりCOMMANDモードに切り替わる。

エディットを終了するには、ED と同様に E コマンドを実行します。

Word Master は、このような感じでエディット作業を行っていく訳ですが、スクリーン・エディット時に多用するカーソルの移動と、スクリーンのスクロールに関するキーの配列を示しておきます。

次の図のように、CTRL キーの近くにダイヤモンド形に配列されていて、慣れると大変具合がいいものです。

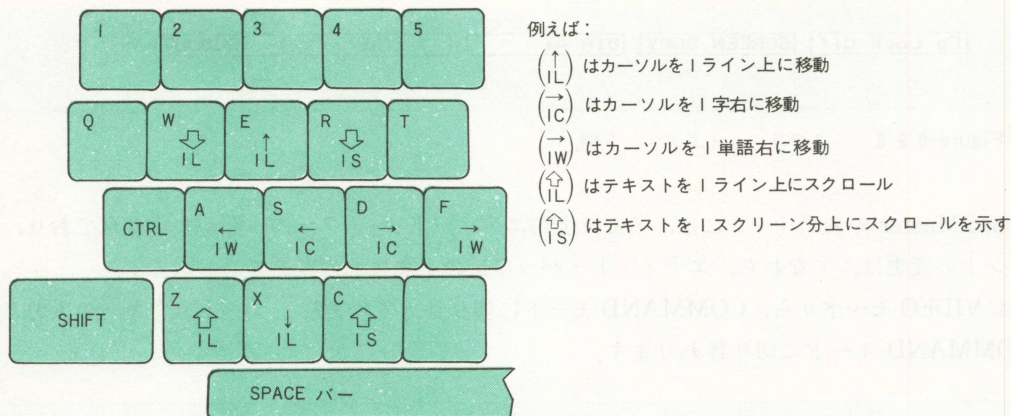


Figure-6.2.8 カーソル移動とスクロール関係のキー

6.3 スクリーン・オリエンテッドなソフトウェアとターミナルの適合について

スクリーン・エディタに限らず、ワード・プロセッサや、6.1章の簡易言語など、スクリーン上に表示されたそれぞれの位置を対象としています。このように、いろいろな操作を行うスクリーン・オリエンテッドな（そのターミナルやコンピュータでしか動かない）アプリケーション・プログラムの実行には、それぞれのコンピュータに、各種のスクリーン・コントロールの機能が必要です。

スクリーン・コントロール機能の主なものを列記すると、

- 画面全体のクリア、
- カーソルをホーム位置（左上端）にセット、
- カーソルのアドレッシング（任意の位置にセット）、
- カーソル位置から、そのラインの終わりまでのイレーズ、
- カーソル位置から、画面最後まででのイレーズ、
- 輝度のコントロール、

などであり、これらは、一般的なエスケープ・シーケンスなどでコントロール可能でなければなりません。

6.3.1 エスケープ・シーケンスとは

エスケープ・シーケンスとは、エスケープ・コード（1BH）に続くアスキー・キャラクタの組み合わせをシーケンシャルにスクリーンに出力することにより、上記の各機能のコントロールなどを、そ

それぞれの組み合わせに従って行わせるものです。

例えば、アメリカで大変ポピュラーな CRT ターミナルである、SOROC IQ-120 のエスケープ・シーケンスの実例のいくつかを示しますと、

- スクリーンのオールクリア.....[ESC]* (つまり 1 BH, 2 AH)
- カーソル位置からその行の終わりまでをイレース...[ESC]T (つまり 1 BH, 54H)
- カーソル位置から画面終わりまでをイレース.....[ESC]Y (つまり 1 BH, 59H)
- カーソルのアドレッシング.....[ESC]=(y)(x) (つまり 1 BH, 3DH, yyH, xxH)

という具合です。

これを具体的に説明すると、スクリーンのオール・クリアの場合は、エスケープ・コードの 1 BH, それに "*" のアスキー・コードである 2 AHを、1 BH, 2 AHと連続してコンソールに出力することにより、コンソールは、それをスクリーンのオール・クリアであると解釈し、実行します。

もう一例として、スクリーン上のX=30, Y=10のポイント、つまり、アドレス (30, 10) のポイントにカーソルをセットする場合は、1 BH, 3 DH, 2 AH(=32+10), 3 EH(=32+30) の4バイトを連続してコンソールに出力すれば良いのです。SOROC IQ-120のカーソルのアドレッシングのX, Y座標には、それぞれ32(=20H)のバイアス値が加算されます。つまり、原点は左上端であり、その座標 (X, Y) は (0, 0) ではなく (20H, 20H) である訳です。

また、スクリーン・コントロールは、これらのエスケープ・シーケンスによるものだけでなく、コントロール・キャラクタによるものも併用されるのが普通です。

上記は、SOROC IQ-120 CRT ターミナルの例ですが、エスケープ・シーケンスを、どのようなシーケンスにするかは、残念ながら統一規格というものはなく、ターミナルのメーカーによってまちまちです。

そこで、ユーザーは、それぞれ自分のターミナルに合わせて、使用するソフトウェアのスクリーン出力部のルーチンを書き替えなければならないことになります。

6.3.2 自分のターミナルに適合させるためのスクリーン出力部の変更

スクリーン・コントロールのコントロール法は統一されておらず、ユーザーは、次の実行例にも示されているような、いろいろなターミナルを使用しています。しかし、ユーザーが、それぞれ自分の使用しているターミナルに合わせて、スクリーン・オリエンテッドなソフトウェアのスクリーン出力ルーチンを、いちいち書き直すのでは大変です。

そこで、スクリーン・エディタや、簡易言語や、他のスクリーン・オリエンテッドなソフトウェア製品の多くは、それぞれのターミナルに簡単に適合させるための特別なプログラムを用意しています。

その1つを、実例として示します。

6.1章で紹介した簡易言語の SuperCalc は、スクリーン表示をフルに活用したソフトウェアであり、その実行には、各種のスクリーン・コントロールが必要です。それらのスクリーン・コントロールの機能を持たない CP/M マシンでは、実行することはできません。

SuperCalc には、いろいろなターミナルに自動的に適合させるためのプログラム "INSTALL.COM" が付属しており、誰でも極めて簡単に自分の使っているターミナルに適合させることができます。

次にこのプログラムを起動して、途中の設問に答えた状態のスクリーンの表示の一部を示します。

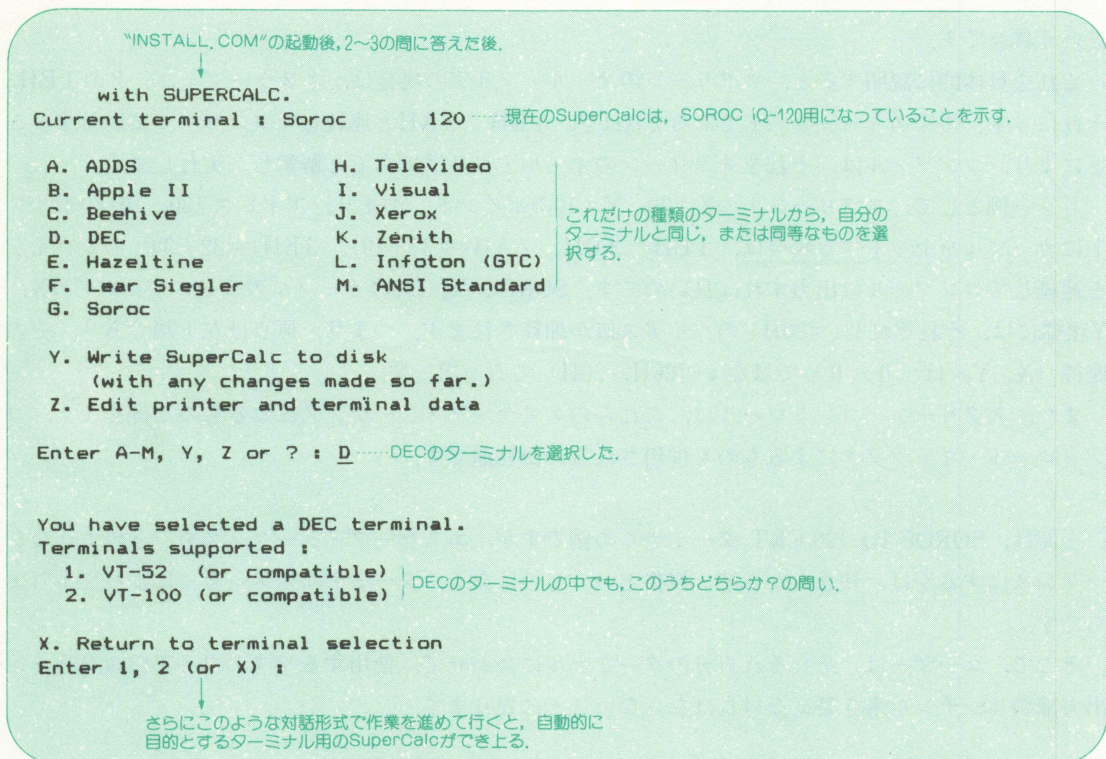


Figure-6.3.1 SuperCalc の各種ターミナルに適合させるための "INSTALL" プログラムの実行の一部。

このように、やさしい対話形式なので、専門の知識を持っていなくても誰でも簡単に自分のターミナルに適合させることが可能です。Apple IIにも適合できますので、Apple ユーザーはお見逃しなく。ただし、マイクロソフト社の Z80 ソフトカードによる CP/M 上での話です。

もし、リストアップされている、どのターミナルとも同等でない場合は、DDT などを利用して、直接、該当部分のルーチンを書き替えることになります。

以上は、SuperCalc の各種ターミナルに適合させるプログラムの一例ですが、他のスクリーン・オリエンテッドなソフトウェアも、まともな製品ならば、ほとんどがこのような便利なプログラムを付属させています (MultiplanやCIS-COBOLのものは特に良くできている)。

このように、最近のソフトウェアは、スクリーン・オリエンテッドなものが多くなり、CP/M マシンのターミナルは、各種スクリーン・コントロールがサポートされているものを使うことが必須となっています。

6.3.3 パーソナル・コンピュータの場合

パーソナル・コンピュータで、このようなスクリーンの機能が最初からサポートされているものは少なく、しかたなく CP/M の BIOS 内に組み込まなければなりません。非常にうまく BIOS 内にスクリーン・コントロール機能を組み込んだ例としては、NEC の PC-8000 シリーズ用に NEC が発売している CP/M や、沖の if800CP/M などがお手本です。

これは、BIOS 内に、前述の SOROC IQ-120 と等価のスクリーン・コントロール・ルーチンを組み込んだもので、ユーザーは、PC-8000 シリーズのパーソナル・コンピュータのスクリーンを SOROC IQ-120 と思って使用することができ、各種のスクリーン・オリエンテッドなソフトウェアを自由に実行することが可能です。

よって、パーソナル・コンピュータの CP/M を購入する際は、この点もよく確認しなければなりません。非常に重要なポイントです。

6.4 CP/MマシンとPROM書込器との接続

CP/M 上で開発したプログラムを PROM に固定したい場合、あるいは、すでに書き込まれている PROM 上のデータを読み出して、逆アセンブラなどで解析したい場合などの、CP/M マシンと PROM 書込器との間のデータ転送の実例を紹介しましょう。

とは言っても、このようなことは一項を設けて解説するまでもなく、PIP コマンドを使いこなせるユーザーであれば、造作もないことなのですが、RS-232C インターフェイスは、たった3本の線で接続するだけで相互に通信可能であることなどを含めて、実例で示しましょう (PIP コマンドのすべての使い方は、「実習 CP/M」の4.2章で解説しています。参照下さい。)

使用する PROM 書込器には、RS-232C のインターフェイスが装備されており、インテル HEX 形式のデータの送受ができるソフトウェアが内蔵されていることが前提となります。大抵の本格的な PROM 書込器は、このような機能を持っています。

一方、パーソナル・コンピュータでなく、ソフトウェア開発用のマシンは、PROM 書込器が内蔵されているのも多く、独立した書込器を使う必要はありませんので、ここでの解説からは除外します。

6.4.1 RS-232C インターフェイスの接続

RS-232C インターフェイスのコネクタの一般的な標準は、25ピンのD型コネクタで、次のピン・アサイメントになっています。

1.	CG (chasis GND)	ケースの GND
← 2.	TD (Tx Data)	送信データ
→ 3.	RD (Rx Data)	受信データ
← 4.	RTS (Request To Send)	送信要求
→ 5.	CTS (Clear To Send)	送信可
→ 6.	DSR (Data Set Ready)	データ送受可
7.	SG (Signal Ground)	信号 GND
→ 8.	CD (Carrier Detect)	キャリア検出
← 20	DTR (Data Terminal Ready)	データ送受可

⋮

矢印の向きは、コネクタから見た信号の流れ。

上記のピン・アサイメントは、一応の標準ではありますが、統一されている訳ではありませんので、各社まちまちなのが現状です。

そこで、CP/M マシンと PROM 書込器との RS-232C インターフェイス相互の接続は、それぞれのハードウェア・マニュアルに従って行わなければなりません。



PC-8001のCP/Mに接続されたPROM書込器

RS-232C の動作は、各種コントロール・ラインによって、コントロールされて、高速のシリアル伝送が可能となりますが、300ボー以下程度の速度であれば、コントロール・ラインによる制御を受けずにデータの送受が可能です。

300ボーの低速では、PROM 書込器を頻繁に使うユーザーにとっては、仕事にならないと思われるので、きちんとハンドシェイクを行って高速伝送が可能となるように接続して下さい。この仕事は、それぞれのインターフェイスのマニュアルが不備であったりして、なかなかうまく行かないものです。

ここでは、次の図に示すように、コントロール・ラインを全く相互に接続せず、GND と Tx, Rx の3線で接続し、300ボー以下で使用することにして、PROM書込器との転送作業を行ってみましょう。この方式ですと、大抵の RS-232C 機器間で無条件に伝送が可能です（インターフェイスによっては、自分自身のコネクタ内で、コントロール線をジャンパする必要があるかも知れません）。

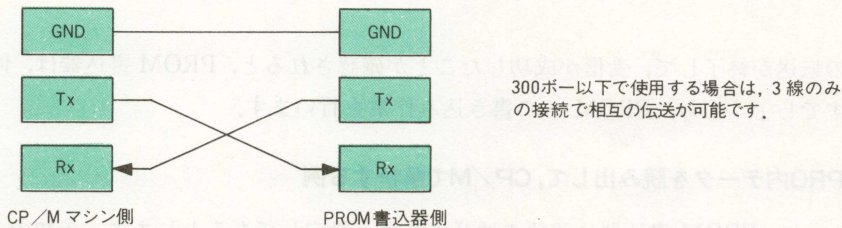


Figure-6.4.1 RS-232Cを3線で相互接続

6.4.2 CP/Mで開発したプログラムのPROM書込器への転送

両者の間は、Figure-6.4.1に示されているように接続されており、ボーレートは両者共に300ボーに設定されているとします。

開発されたプログラムは、インテル HEX 形式のファイル（ROM. HEX とでもしておく）でディスク上にあるとして、2 Kバイト未満のオブジェクトで2716タイプの PROM に書き込む場合の手順を示します。

1. まず、PROM 書込器のマニュアルに従って、RS-232C インターフェイスからデータを受け取る準備をし、用意ができたなら受信状態にしておきます。
2. PIP コマンドで転送するファイル "ROM. HEX" を RS-232C から送り出します。PIP コマンドのデバイス名は、コンピュータによって異なります。ここでの実例は、PC-8801で NEC の CP/Mを使った場合のものです。その様子を次に示します。

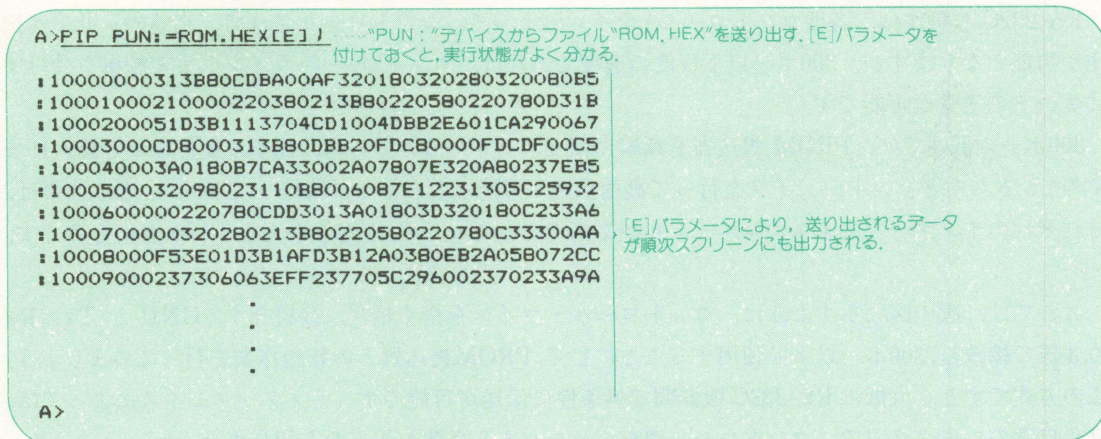


Figure-6.4.2 CP/M マシンからでき上ったプログラムを PROM 書込器に転送する。

- データの転送が終了して、受信が成功したことが確認されると、PROM 書込器は、何らかの合図を出すでしょうから、PROM への書き込み作業を行います。

6.4.3 未知のPROM内データを読み出して、CP/Mで解析する例

CP/M マシンと、PROM 書込器は前項の通りに接続・設定してあるとします。未知の PROM には、前項で書き込んだばかりのチップをサンプルとしましょう。

PROM 内のデータを CP/M マシンで受信して、HEX ファイルとしてディスクにセーブする手順を示します。

- まず、CP/M マシン側で、先に PIP コマンドを実行しておきます。ファイル名は "ROM1 . HEX" とでもしておきました。

A>PIP ROM1 . HEX=RDR: [E]]

PIP コマンドのデバイス名は、コンピュータによって異なりますが、ここでの実例は、PC-8801 での NEC の CP/M のものです。

- この後に、PROM 書込器を操作して、未知の PROM 内のデータを読み出して、RS-232C から HEX コードとして送り出します。

その時の CP/M 側での受信例と、生成された HEX ファイルの確認の様子を次に示します。

A>PIP ROM1.HEX=RDR:[E]

“RDR:”デバイスから受信して、ファイル“ROM1.HEX”を作る。[E]/パラメータを付けておくと、受信状態がよく分かる。さらに[H]/パラメータを付けて“HE”とする方がベターである。HEXファイルのエラーのチェックを行ってくれる。

```
10000000313BB0CDBA00AF3201B03202B0320080B5
100010002100002203B0213BB02205B02207B0D31B
1000200051D3B1113704CD1004DBB2E601CA290067
10003000CD8000313BB0DBB20FDCB0000FDCDF00C5
100040003A01B0B7CA33002A07B07E320AB0237EB5
100050003209B023110BB0060B7E12231305C25932
10006000002207B0CDD3013A01B03D3201B0C233A6
10007000003202B0213BB02205B02207B0C33300AA
10008000F53E01D3B1AFD3B12A03B0EB2A05B072CC
10009000237306063EFF237705C296002370233A9A
```

[E]/パラメータにより、受信されるデータが順次スクリーンにも出力される。

```
1007D000FF00FF10FF00FF0000FF00FF00FF00FF11
1007E000FF00FF00FF00FF0000FF00FF00FF00FF11
1007F000FF006F00BF00FF00B4FF00FF50FF00FFFD
10080001F7
```

A>データの終わりを受信すると、自動的にファイル“ROM1.HEX”が作られる。

Figure-6.4.3 PROM 書き込器からの未知の ROM データの受信。

A>STAT ROM1.HEX]生成されたファイルの確認。

```
RECS  BYTES  EXT  ACC
  46      6K    1  R/W  A:ROM1.HEX .....6K/バイトのHEXファイルが作られている。
BYTES REMAINING ON A: 14K
```

A>

Figure-6.4.4 生成された HEX ファイルの確認。

- 生成された未知の ROM 内データのファイル“ROM1.HEX”を DDT や ZSID などにより解析します。DDT によりメモリ上にロードした様子を次に示します。
この ROM 内データは、Figure-6.4.4のHEX データのアドレス部から、ロード・アドレスは 0 Hであることが分かるので、DDT などでメモリ上にロードする場合は、100H以上にロードするためのバイアスをかけなければならないことに注意します。


```

A>DDT /      DDTを起動.

DDT VERS 2.2
-IROM1.HEX /  100Hのバイアスをつけて"ROM1.HEX"をロード.
-R100 /        (100Hのバイアスをつけたのは、このプログラムが0H
NEXT PC       スタートのため)
0900 0800

-L100 /
0100 LXI SP,803B
0103 CALL 00BA
0106 XRA A
0107 STA 8001
010A STA 8002
010D STA 8000
0110 LXI H,0000
0113 SHLD 8003
0116 LXI H,803B
0119 SHLD 8005
011C SHLD 8007
.
.
.
.

```

未知のROM内プログラムが、CP/Mのメモリ上にロードされた。以後は逆アセンブルなどで解析を行って行く。

Figure-6.4.5 未知のROM内データを、CP/M上で解析する。

6.5 CP/Mマシン間の音響カプラによる通信

最近、数社から5万円を切る低価格の音響カプラが発売されており、電話がつながる所なら日本国内と言わず世界中どこでもCP/Mマシン間のデータの送受が行えます。

音響カプラのほとんどのものは、RS-232Cインターフェイスで入出力を行っているので、CP/Mマシンは、RS-232Cインターフェイスがサポートされているものであることが前提となります。

音響カプラによる電話線を介してのデータの伝送は、各種ノイズが混入する機会が非常に多く、長いデータ伝送になればなるほど、エラー率は増加します。よって、電話線を介しての伝送は、エラーは必ず発生することを考慮して作られたプロトコルを持った「データ通信用プログラム」によってデータ伝送を行うことが不可欠です。

そのプロトコルの基本は、例えば、送り出し側で128バイトとか、256バイトとかを1ブロックとし、ブロック単位でチェックサムやCRC(Cyclic Redundancy Check：チェックサムより高度なエラー・チェック方式)などを算出し、そのブロックの最後に付加して送り出します。受信側では、受信したデータをブロック単位でチェックし、OKであればOKサインを送信側に返し、次のブロックの送信を要求します。もしエラーが検出されれば再び同じブロックの再送信を要求します。

と、このような手順で送受両者がハンドシェイクしながらデータの送受を行う訳です。

6.5.1 専用プログラムを使用せず、PIPコマンドで伝送する例

通信プロトコルを持った専用プログラムを使用しなければ、電話線を介しての伝送は実用にはなりません。しかし、CP/MのPIPコマンドには、HEXファイルの送受の際の異状を検出する[H]パラメータがあり(実習CP/M 4.2章参照)、短いインテルHEX形式のデータなら、伝送可能です。ただし、PIPはエラーを検出すると自動的に入力を停止しますので、その場合は最初からやり直さなければなりません。

では、CP/Mのマスター・ディスクに含まれている“DUMP.ASM”を、ASMでアセンブルして生成される“DUMP.HEX”を音響カプラを使って、電話線で伝送する例を示します。

1. 6.4章の場合と同様に、CP/Mマシンと音響カプラのRS-232Cインターフェイスを接続し、ボーレートを300ボーに設定します。音響カプラの通信モードは、送受とも半二重(HALF)にセットします。CALL→ANSの切替は、送信側が“ANS”、受信側が“CALL”にセットします。
2. 相手と電話で打ち合わせをし、その後、受話器を音響カプラにセットし、カプラの電源を入れます。
3. まず受信側から先に次のPIPコマンドを実行します。作られるファイル名は、ここではTEST.HEXとでもしておきます。

```
A>PIP TEST.HEX=RDR:[HE] J
```

4. 手順3.のあとに、送信側は次のPIPコマンドを実行します。

```
A>PIP PUN:=DUMP.HEX[E], EOF: J
```

5. ノイズなどによるエラーがなければ、これで自動的に受信したデータのファイルが作られます。エラーが発生すれば、受信側ではエラー・メッセージを出力して、受信を停止するので、再び手順の2.からやり直します。

次に受信側の実行の様子を示します。送信側は、PIPのコマンド・ラインが異なるだけで同様なので省略します。


```

A>PIP TEST.HEX=RDR:[HE]! .....RDR:「デバイスから入力して、そのデータをファイル「TEST.HEX」としてセーブする。
    HEXファイルをチェックをさせるための[H]パラメータを必ず付けること。

:1001000021000039221502315702CDC101FEFFC284
:100110001B0111F301CD9C01C351013E803213023A
:10012000210000E5CDA201E1DA5101477DE60FC2D1
:100130004401CD7201CD59010FDA51017CCD8F01FF
:100140007DCD8F01233E20CD650178CD8F01C32366
:1001500001CD72012A1502F9C9E5D5C50E0BCD05F1
:1001600000C1D1E1C9E5D5C50E025FCD0500C1D101
    .
    .
    .
:1001E000452044554D502056455253494F4E2031DD
:1001F0002E34240D0A4E4F20494E50555420464966
:100200004C452050524553454E54204F4E204449B2
:03021000534B2429
:0000000000

[E]パラメータにより、入力データは、順次
    スクリーンにも表示される。

A> .....エラーなく受信できた場合には、以上のHEXデータが、ファイル「TEST.HEX」としてセーブされている。
    
```

Figure-6.5.1 音響カプラによる電話線を使つてのデータ受信側。

注) PIP コマンドのデバイスは、コンピュータによって異なります。ここでの例は、送受とも PC-8801の RS-232C インターフェイスを、NEC の CP/M で使用したものです。

以上のような手順で、専用の通信プログラムがなくても、HEX ファイルであれば、2～3回やり直しをすれば伝送可能です。通常は、通信プログラムを使っていますので、今回の方法は初めてのことであり、アスキーの出版部と、筆者宅の間で実験を行いました。回線の状態に大きく依存しますが、だいたい2回に1回は成功するようです。



音響カプラによる通信を行っている筆者の現在のマシン類。電話機の左はミニファクス。右端は、YD-74C×2 ドライブを接続した往年のS-100システム SOL-20。(本書が出版される頃、左端の8インチシステムで満足のいく高速CP/M(未発売)が稼動を始め、随分と活躍したSOLも交代の時代が来た。)

あとがき

アスキー・ラーニングシステムの CP/M シリーズは、この「応用 CP/M」で一応完結です。「入門 CP/M」に着手してから、この応用編が発売されるまで、1 年半もかかってしまいました。しかし、なんとかまとめ上げることができたのは、「早く応用 CP/M を出せ」という実に多くの当シリーズ愛読者の多大な期待に支えられていたためです。どうもありがとうございました。

CP/M の良さは、その簡潔さにあります。その簡潔であるために持ち合せていない機能を捕らえて、欠点であると言う人も中にはいます。私はそれらの人は、もしかしたら 8 ビットのマイクロコンピュータに取り組んだことがないのではないかと思います。相手はミニコンピュータや大型コンピュータではありません。

要はコンピュータの能力とのバランスのとれた OS であることが大切です。16 ビットの、それも 68000 あたりのコンピュータであれば、現在の CP/M では明らかにバランスしません。そこでは新しい CP/M であるコンカレント CP/M とか、UNIX とかの世界になるでしょう。

私は、当シリーズの最後に、「応用 CP/M」を読まれた読者のみなさんに 1 つお願いがあります。

それは、CP/M の存在を知らなかったり、名前は知っていても取りかかる糸口が見つからないでいる多くの優秀な若者に、この CP/M の世界があることを教えてあげてほしいのです。彼らの多くは、パーソナル・コンピュータに組み込まれている BASIC 言語の殻の中だけでコンピュータというものを見ていると思うのです。しかし何かのきっかけで、アセンブラなり他の言語なりに興味を持つようになり、大きく飛躍することができるかも知れません。BASIC から離れようとした時、そこからがすべての始まりであるという気がしてならないのです。

読者の回りに、そのような熱心な若者がいたら、是非 CP/M の世界のことを話してあげて下さい。

ソフトウェアの世界は限りなく広く、CP/M はこの海に乗り出すための小さな船に過ぎません。しかし、小さいながら信頼できる船です。

今夜は、シンプルで小さな船「CP/M」に乾杯！

5 章の「各種高級言語」では、ABCの河田至弘氏、電通大の打越浩幸氏、リギーコーポレーションの片桐明氏にいろいろと助けを頂きました。無理なお願いを心良く引き受けて下さって、ほんとうにありがとうございました。

村瀬 康治
テレビ朝日技術局勤務
初版時 36歳

付録A NECのCP/Mについて

1982年7月に、PC-8001またはPC-8801に共通のCP/MがNECから発売されました。ディスクのアクセス・スピードは、8インチの標準ディスクの場合より速く非常に高速であり、日本のCP/Mでは初めての“タイプ・アヘッド機能”など、多くの拡張機能を持った画期的なものです。ここではNECのCP/Mで一番最初に発売された、ミニディスク用片面バージョンのものの概要を紹介しておきます。

●対象機種：

PC-8001+ (PC-8011またはPC-8012・8013) および、
PC-8801 (N-BASIC モードで動作)。

●使用ディスク・ユニット：

PC-8031-1WあるいはPC-8031-2W (ただし1Wとして動作する) のいずれでも可。

●NEC-CP/M 独自の拡張機能：

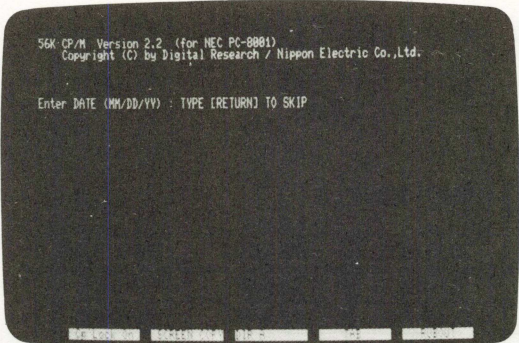
- 1) エスケープ・シーケンス等によるスクリーン・コントロール(6.3章参照), およびN-BASIC内グラフィック機能などのコントロール,
- 2) N-BASIC ROM 内各ルーチンの呼び出し機能,
- 3) 割り込み処理によるバッファリング・キー入力機能 (これを“タイプ・アヘッド機能”と言いい、コンピュータがいかなる処理を行っている場合でも、また、いかに速くタイプインしようと、キーが押されたものは1文字たりとも取りこぼすことなく入力が行える機能)。
- 4) 任意のプログラムを、CP/Mの起動時にオート・スタートさせることができる。
- 5) 割り込み処理によるRS-232Cポートを2チャンネル、通常処理によるものを1チャンネル、計3チャンネルのRS-232Cポートをサポート。
- 6) ファンクション・キーによりスクリーン・コピー等の機能をサポートし、ユーザーによる定義も可能。
(その他省略)

● マスター・ディスクに含まれている独自のユーティリティ・プログラム：

- 1) ディスク・フォーマット.
- 2) ディスク・コピー.
- 3) オート・スタート設定プログラム.
- 4) ファンクション・キー定義プログラム.
- 5) CP/M サイズの変更を、完全に自動的に行うプログラム.

(その他省略)

NECは、このミニディスクの1W用のあと、2W用のCP/M、それにPC-8801 N88モードで動作する8インチ・ディスク用CP/Mを順次発売していく予定でいるようです。期日は未定。



PC-8001 CP/Mのオープニング画面

A>STAT *.*

Recs	Bytes	Ext	Acc
64	8k	1	R/W A:ASM.COM
8	1k	1	R/W A:AUTHELLO.COM
6	1k	1	R/W A:AUTO-ST.COM
18	3k	1	R/W A:COPY.COM
38	5k	1	R/W A:DDT.COM
49	7k	1	R/W A:DISKDEF.LIB
33	5k	1	R/W A:DUMP.ASM
3	1k	1	R/W A:DUMP.COM
52	7k	1	R/W A:ED.COM
10	2k	1	R/W A:FORMAT.COM
143	18k	2	R/W A:HELLO.ASM
32	4k	1	R/W A:HELLO56.COM
29	4k	1	R/W A:KEY.COM
14	2k	1	R/W A:LOAD.COM
32	4k	1	R/W A:MODULE.HEX
80	10k	1	R/W A:MOVCPM.COM
58	8k	1	R/W A:PIP.COM
41	6k	1	R/W A:STAT.COM
10	2k	1	R/W A:SUBMIT.COM
4	1k	1	R/W A:SWITCH.COM
7	1k	1	R/W A:SYSGEN.COM
66	9k	1	R/W A:USER.ASM
101	13k	1	R/W A:USERIO.ASM
32	4k	1	R/W A:WCLOCK.COM
6	1k	1	R/W A:XSUB.COM
Bytes Remaining On A: 3k			

A>

Figure-A.1 PC-8001 CP/Mに含まれているファイル。

313

付録B CP/M上で走るBASIC言語の ステートメント・関数比較一覧表

CP/M上の高級言語で、最もポピュラーに使われている BASIC 言語の代表的なものを 4 種類（そのうち 2 種はコンパイラ）を取り上げ、そのステートメントと関数の一覧表を示します。

取り上げる言語を次に示します。

- MBASIC（インタープリタ）
- BASCOM（コンパイラ、3.4 章参照）
- CBASIC（中間コード・インタープリタ）
- CB-80（コンパイラ、5.3 章参照）

1. ステートメントおよび関数一覧表

分類	機 能	CBASIC	CB-80	MBASIC	BASCOM
実行制御	分割したプログラムをディスクから順次ロードし実行	CHAIN	CHAIN	CHAIN	CHAIN
	別のプログラム・モジュールへの変数の引渡し	COMMON	COMMON	COMMON	COMMON
	SUB 手続きに処理を移行		CALL		
	機械語のプログラムの実行開始	CALL	CALL	CALL	CALL
入出力ステートメント (1) ファイル以外	キーボードから入力して変数に代入	INPUT	INPUT	INPUT	INPUT
	コンマもデータとしてキーボードから 1 行入力	INPUT LINE	INPUT LINE	LINE INPUT	LINE INPUT
	画面に情報を出力	PRINT	PRINT	PRINT	PRINT
	画面に指定書式で文字、数値を表示	PRINT USING	PRINT USING	PRINT USING	PRINT USING
	プリンタに情報を出力	PRINT	PRINT	LPRINT	LPRINT
	プリンタに指定書式で文字、数値を印字	PRINT USING	PRINT USING	PRINT USING	PRINT USING
	指定した出力ポートにデータを出力	OUT	OUT	OUT	OUT
	メモリの指定番地にデータを書き込む	POKE	POKE	POKE	POKE
	出力装置をコンソールからプリンタに変更	LPRINTER	LPRINTER		
	出力装置をプリンタからコンソールに変更	CONSOLE	CONSOLE		
	DATA 文で定義された定数を変数に代入	READ	READ	READ	READ
	READ 文で読み込まれるデータの格納	DATA	DATA	DATA	DATA
	DATA 文の値の再読み込み指定	RESTORE	RESTORE	RESTORE	RESTORE
入出力ステートメント (2) ファイル	シーケンシャル・ファイルからのデータ入力	READ #	READ #	INPUT #	INPUT #
	ランダム・ファイルからのデータ入力	READ #	READ #	GET #	GET #
	シーケンシャル・ファイルからの 1 行入力	READ # LINE	READ # LINE	LINE INPUT #	LINE INPUT #
	ランダム・ファイルからの 1 行入力	READ # LINE	READ # LINE		

分類	機 能	CBASIC	CB-80	MBASIC	BASCOM
入出力ステートメント (2)	ディスクから1バイトのデータを入力		GET		
	シーケンシャル・ファイルにデータを出力	PRINT #	PRINT #	PRINT #	PRINT #
	ランダム・ファイルにデータを出力	PRINT #	PRINT #	PUT #	PUT #
	シーケンシャル・ファイルに指定書式でデータを出力	PRINT # USING	PRINT # USING	PRINT # USING	PRINT # USING
	ランダム・ファイルに指定書式でデータを出力	PRINT # USING	PRINT # USING		
	ディスクに1バイトのデータを出力		PUT		
分岐ステートメント	指定された行番号の文へ無条件分岐	GOTO	GOTO	GOTO	GOTO
	指定されたラベルの文へ無条件分岐		GOTO		
	指定された行番号からのサブルーチンへ分岐	GOSUB	GOSUB	GOSUB	GOSUB
	指定されたラベルのサブルーチンへ分岐		GOSUB		
	条件判断によるプログラムの流れの制御	IF~THEN~ELSE	IF~THEN~ELSE	IF~THEN~ELSE	IF~THEN~ELSE
	式の値により指定行番号の文へ多岐分岐	ON~GOTO	ON~GOTO	ON~GOTO	ON~GOTO
	式の値により指定ラベルの文へ多岐分岐		ON~GOTO		
	式の値により指定行番号からのサブルーチンへ分岐	ON~GOSUB	ON~GOSUB	ON~GOSUB	ON~GOSUB
	式の値により指定ラベルのサブルーチンへ分岐		ON~GOSUB		
ループ	一連の命令を指定回数繰り返し実行	FOR~NEXT	FOR~NEXT	FOR~NEXT	FOR~NEXT
	FOR~NEXT 文の増分を指定	STEP	STEP	STEP	STEP
	一連の命令を条件つきで繰り返し実行	WHILE~WEND	WHILE~WEND	WHILE~WEND	WHILE~WEND
ファイル管理ステートメント	ファイルの新規作成	CREATE	CREATE	OPEN	OPEN
	ファイルの抹消	DELETE	DELETE	KILL	KILL
	ファイル名の変更	RENAME	RENAME	RENAME	RENAME
	ファイルを削除変更から保護		LOCK		
	ファイルの保護を解除		UNLOCK		
	ファイルバッファの割り当て	OPEN	OPEN	OPEN	OPEN
	ファイルバッファの解除	CLOSE	CLOSE	CLOSE	CLOSE
定義ステートメント	プログラム中の注釈指定	REM,REMARK	REM,REMARK	REM	REM
	配列次元数, 添字の指定	DIM	DIM	DIM	DIM
	配列の最小添字の指定			OPTION BASE	OPTION BASE
	機械語プログラムのためのメモリ確保	SAVEMEM	SAVEMEM	CLEAR	CLEAR
	ユーザー関数の定義(1行)	DEF FN	DEF	DEF FN	DEF FN
	ユーザー関数の定義(複数行)	DEF FN~FEND	DEF~FEND		
	関数型SUB手続の定義		DEF~FEND		
	変数の型を整数として宣言	INTEGER	INTEGER	DEFINT	DEFINT
	変数の型を半精度実数として宣言			DEFSNG	DEFSNG
	変数の型を全精度実数として宣言	REAL	REAL	DEFDBL	DEFDBL
	変数の型を文字として宣言	STRING	STRING	DEFSTR	DEFSTR
	乱数系列の変更	RANDOMIZE	RANDOMIZE	RANDOMIZE	RANDOMIZE
	コンソール1行あたりの表示数の指定	POKE 272,n		WIDTH	WIDTH
	プリンタ1行あたりの印字数の指定	WIDTH		WIDTH LPRINT	WIDTH LPRINT
特殊ステートメント	エラー時処理ルーチンの指定		ON ERROR GOTO	ON ERROR GOTO	ON ERROR GOTO
	ファイルの終了を検出		IF END #	EOF	EOF
	プログラム実行状態の追跡	TRACE		TRON	TRON
	プログラム実行状態の追跡の解除			TROFF	TROFF

CP/Mで走るBASIC言語の
ステートメント・関数比較一覧表

分類	機 能	CBASIC	CB-80	MBASIC	BASCOM
文字型 スト リング 関数	引数の数値を文字列に変換	STR\$	STR\$	STR\$	STR\$
	アスキーコードを文字に変換	CHR\$	CHR\$	CHR\$	CHR\$
	指定文字位置から任意字数の取り出し	MID\$	MID\$	MID\$	MID\$
	文字列の左から指定数の文字の取り出し	LEFT\$	LEFT\$	LEFT\$	LEFT\$
	与えられた文字の繰り返し文字列の作成			STRING\$	STRING\$
	指定数のスペースからなる文字列の作成			SPACE\$	SPACE\$
	文字列中の小文字を大文字に変換	UCASE\$	UCASE\$		
	10進数を16進の文字列に変換			HEX\$	HEX\$
	10進数を8進の文字列に変換			OCT\$	OCT\$
数スト リ ン グ 関 数 型	文字列の最初の文字をアスキーコードに変換	ASC	ASC	ASC	ASC
	指定文字列を探し出し、その位置を与える	MATCH	MATCH	INSTR	INSTR
	文字列の長さを与える	LEN	LEN	LEN	LEN
	数字列を数値に変換	VAL	VAL	VAL	VAL
入出力 関数 他	指定ポートから1バイト読み込む	INP	INP	INP	INP
	メモリの指定番地の内容を与える	PEEK	PEEK	PEEK	PEEK
	コンソール・ステータスを与える	CONSTAT%	CONSTAT%		
	コンソール・デバイスから1文字読み込む	CONCHAR%	CONCHAR%		
	CP/Mコマンド行の文字列を与える	COMAND\$	COMAND\$		
	変数に割り付けられたメモリ番地を与える	VARPTR	VARPTR	VARPTR	VARPTR
	文字列に割り付けられたメモリ番地を与える	SADD	SADD		
	指定ファイルのブロック数を与える	SIZE	SIZE		
	エラー・コードを与える		ERR	ERR	ERR
	エラーの発生した行番号を与える		ERL	ERL	ERL
	コンソールから1文字読み込みエコーバックしない		INKEY	INPUT\$	INPUT\$
	コンソールに出力される、次の水平位置を与える	POS	POS	POS	POS
	プリンタに印字される、次の水平位置を与える	POS	POS	LPOS	LPOS
数値 関数	絶対値 [X]	ABS(x)		ABS(x)	
	切捨て [X] (返される値は実数)	INT(x)		INT(x)	
	剰余 $X - [X/m] \times m$	MOD(x,m)		x MOD m(演算子)	
	整数の実数化	FLOAT(x)		CDBL(x), CSNG(x)	
	実数の整数化	INT%(x)		CINT(x)	
	文字列の数値化	VAL(x)		VAL(x)	
	正負符号の取り出し	SGN(x)		SGN(x)	
数 値 関 数	半精度定数の全精度実数化			CDBL(x)	
	全精度実数の半精度実数化			CSNG(x)	
	指数関数 e^x	EXP(x)		EPX(x)	
	自然対数関数 $\log X$	LOG(x)		LOG(x)	
	正弦関数 $\sin X$	SIN(x)		SIN(x)	
	余弦関数 $\cos X$	COS(x)		COS(x)	
	正接関数 $\tan X$	TAN(x)		TAN(x)	
	逆正接関数 $\arctan X$	ATN(x)		ATN(x)	
	平方根 \sqrt{X}	SQR(x)		SQR(x)	
	乱数	RND		RND	

2. 定数, 変数 仕様一覧表

分類	機 能		CBASIC, CB-80	MBASIC, BASCOM
定 数	整 定 数	最大	+32767	+32767
		最小	-32768	-32768
	半精度実定数	有効桁数		7 桁
		最大指数		38
		最小指数		-38
	全精度実定数	有効桁数	14桁	16桁
		最大指数	62	38
		最小指数	-64	-38
	進法表示	16進定数	数値の後に H	数値の前に &H
		8 進定数		数値の前に & または &O
		2 進定数	数値の後に B	
	文字定数	最大文字列定数表	255文字	255文字
変 数	属性文字	整数型	%	%
		半精度実数型		!
		全精度実数型	何もつけない	#
		文字型	\$	\$
	英数字変数名の有効長		31文字	40文字
そ の 他	ラベルの使用		CB-80のみ可	不可
	1 行の最大文字数		255文字	255文字

3. 演算子一覧表

分類	機 能	CBASIC, CB-80	MBASIC, BASCOM
算 術 演 算 子	べき乗	^	^
	負号	-	-
	乗算	*	*
	除算	実数除算	/
		整数除算	\ または %
	剰余	MOD(x,m)(関数)	x MOD m
	加算	+	+
関 係 演 算 子	減算	-	-
	等しい	= または EQ	=
	等しくない	<> または NE	<>
	大きい	> または GT	>
	小さい	< または LT	<
	大きいか等しい	>= または GE	>=
	小さいか等しい	<= または LE	<=
文 字 列 演 算 子	連結	+	+
	等しい	= または EQ	=
	等しくない	<> または NE	<>
	大きい	> または GT	>
	小さい	< または LT	<
	大きいか等しい	>= または GE	>=
	小さいか等しい	<= または LE	<=
論 理 演 算 子	否定	NOT	NOT
	論理積	AND	AND
	(内包的)論理和	OR	OR
	排他的論理和	XOR	XOR
	包含		IMP
	合同		EQV

付録C 本書で使⽤した各種ソフトウェアについて

本書で使⽤したソフトウェアの中で、

MACRO-80 (和⽂マニュアル付)
LINK-80 (和⽂マニュアル付)
BASC0M (和⽂マニュアル付)
FORTRAN-80 (和⽂マニュアル付)
muLISP

以上は、マイクロソフト社の極東総代理店である(株)アスキーコンシューマプログクツ(TEL: 03-486-7111) から、その最新バージョンを快く提供して頂きました。

また、

MAC	PL/I-80 (和⽂マニュアル付)
RMAC	APL\80
ZSID	ACT69
CIS COBOL (和⽂マニュアル付)	XLT86
CB-80	SuperCalc (和⽂マニュアル付)
PASCAL/MT+	UTILITY VOL1 (和⽂マニュアル付)

以上は、デジタルリサーチ社の極東総代理店である、(株)マイクロソフトウェア・アソシエイツ(TEL: 03-497-0381)

から、その最新バージョンを快く提供して頂きました。

また、Rgy FORTH については、

(株)リギーコーポレーション (TEL: 045-313-3038)

から、その最新バージョンの提供と、プログラミングについての親切な助言を頂きました。

これらのソフトウェアの購⼊や、問い合わせについては、上記、または上記の販売店に相談して下さい。参考までに、和⽂のマニュアルが付属しているものは、"(和⽂マニュアル付)"としておきました。

本書で使用した各種ソフトウェアについて

輸入ソフトウェアの和文マニュアル付のものは、上記または上記の販売店によるルートでなければ入手できませんので、よく確認して下さい。

アスキーコンシューマプロダクツの那須勇次氏

マイクロソフトウェア・アソシエイツの社長の岡田純一氏

リギーコーポレーションの社長の片桐明氏

には、当 CP/M シリーズのために何かと協力頂きまして、本当にありがとうございました。



本書で使用したソフトウェアなどのマニュアル

付録D CP/M version 2.2のバグについて

デジタルリサーチ社は、1982年1月に、それ以前にリリースした CP/M version 2.2のバグ情報のまとめを発表しています。

一般的なユーザーには直接関係ないバグもありますので、ここでは、日常使う可能性があるもので、一応は訂正しておいた方がよいと思われるものを2件ほど紹介しておきましょう。それ以後のリリースのものは、除々に修正されつつあると思いますが、次に示すリストの箇所を、一応 DDT で確認してみると良いでしょう。

PIP のパラメータ [S] と [Q] のバグ

PIP のオプションである [S] と [Q] パラメータを使って、[S文字列1^ZQ文字列2^Z]の複合コマンドとして、文字列1から文字列2の間を転送する場合、“文字列1”の字数と“文字列2”の字数が同じ長さの場合、正常な動作をしない場合があります。

これは、次に示す手順でパッチを行うことにより修正できます。

A>DDT PIP.COM / ...DDTを起動し、PIP.COMをロードする。

DDT VERS 2.2

NEXT PC

1E00 0100

-L116B,1179/

116B LDA 1F62

116B STA 1DF7

116E LXI H,1F62

1171 MVI M,00

1173 LDA 1DF9

1176 INR A

1177 STA 1DFB

117A

-A116B/

116B LXI H,1F62/

116B MOV A,M/

116C STA 1DF7/

116F MVI M,0/

1171 LXI H,1DF9/

1174 MOV A,M/

1175 MVI M,0/

1177 INR A/

1178 DCX H/

1179 MOV M,A/

117A ./

-GO/DDTを終わりCP/Mに戻る。

逆アセンブルして、バグ箇所の確認。
このようになっているものは、以下のように修正する。

ライン・アセンブラで、このように修正する。


```
A>SAVE 29 PIP.COM / .....修正したPIP.COMのメモリ・イメージをディスクにセーブする。
A>                                PIP.COMのパッチ完了。
```

Figure-D.1 PIP の [S~Q] パラメータのバグの修正。

SUBMIT ファイルの中のコントロール・キャラクタのバグ

SUBMIT ファイル (.SUB) 中にコントロール・キャラクタを含める場合は、例えば Ctrl-Z であれば、山形印の“^”と“Z”の2文字を使って、“^Z”と書けば、これはSUBMITによってCtrl-Zとして認識されることになっていますが、正常に動作しない場合があります。

これは、次に示す手順でパッチを行うことにより修正できます。

```
A>DDT SUBMIT.COM / .....DDTを起動し、SUBMIT.COMをロード。
DDT VERS 2.2
NEXT PC
0600 0100
-L441 /
  0441 SUI 61
  0443 STA 0E7D
  0446 MOV C,A
  0447 MVI A,19
  0449 CMP C
  044A JNC 0456
  044D LXI B,019D
  0450 CALL 02A7
  0453 JMP 045E
  0456 LDA 0E7D
  0459 INR A
-S442 /
0442 61 41 /
0443 32 . / } アドレス442Hの1/バイトのみ変更する。
-B0 / .....DDTを終りCP/Mに戻る。

A>SAVE 5 SUBMIT.COM / .....修正したSUBMIT.COMのメモリ・イメージをディスクにセーブする。
A>                                SUBMIT.COMのパッチ完了。
```

Figure-D.2 SUBMIT のコントロール・キャラクタのバグ修正。

索引

A

ACT 65	207
ACT 68	207
ACT 69	207
ACT 80	207
ACT 86	207
AIDS	273
ALGOL	221, 274
ALGOL-M	274
ALV	19
AL	20
AL 1	20
ANSI	223
ANSI標準	221
APL	221, 278
APLターミナル・モード	279

B

BASCOM	200
BASIC	221, 234
BASICインタープリタ	199
BASICコンパイラ	199
BDOS	3, 34
BIOS	3, 4, 5
BLM	20
bls	22
BOOT:	11
BSH	20

C

C	221, 259
CBASIC	234
CBIOS	5
CBIOS.ASM	6
CB-80	234
CCP	3
CIS COBOL	223
CKS	20
cks	23
COBOL	221, 222
CODASYL	222
CONIN	10
CONOUT	11
CONOUT:	11
CONST:	11
CRC	308
CSEG	195
CSV	19

C言語	259
-----	-----

D

dir	22
DISKDEFマクロ・ライブラリ	19, 21
DISK.DOC	169
dks	22
DMA	78
DMAバッファ	13, 41, 77
dn	22
DPB	19, 21
DRM	20
DSM	20

E

EXM	20
EXTシンボル	200

F

FCB	34, 77
FCBアドレス	81
FILE	171
FORTH	221, 263
FORTRAN	221, 229
FORTRAN-80	229
fsc	22

G

GENCMD	213
GET	171
global data analysis	213

H

HOME:	12
-------	----

I

IAL	274
I/Oユーティリティ・ライブラリ	252

J

JUMPベクトル	6, 10, 15
----------	-----------

L

LINK	207
LISP	221, 270
LISTST:	13
LIST:	12
lsc	22

M

MAC	22, 165
MAC.COM	169
MACRO-80	186
MP/M	72
MSIZE	6
muLISP	271
Multiplan	287
muSTAR	273

O

OFF	20
ofs	23

P

PAGE	183
PASCAL	221, 240
Pascal/MT+	241
PLMX	252
PL/M	221, 247
PL/I	221, 246
PL/I-80	247
PROM	303
PROM書込器	303
PUNCH	12
PUT	171

R

READER	12
READ	13
Rgy FORTH	264
RMAC	181
RS-232C インターフェイス	304, 308

S

SECTRA	13
SELDISK	6
SELDISK	12
SETDMA	13
SETSEC	12
SETTRK	12
SID	179
skf	22
SOROC IQ-120	301
SPT	20
SuperCalc	287

T

TITLE	183
TPA	3, 40
TRANS	18

U

UNIX	259
------	-----

V

VISICALC	287
----------	-----

W

WBOOT	11
Word Master	294, 295
WRITE	13

X

XLT	19
XLT86	212

Z

ZSID	179
Z80.LIB	165, 174

ア

アクティブ・セル	288
アセンブリ・パラメータ	167
アメリカ国立標準協会	223
アロケーション・アドレス	101
アロケーション・ベクトル	101
インテル型式	186
ウォーム・ブート	11
エスケープ・シーケンス	4, 300
オプション・コマンド・ライン	41
音響カブラ	308
オンライン	97

カ

簡易言語	287
疑似命令	207
クロス・アセンブラ	207
高級言語	199
コールド・スタート・ローダ	11
コールド・ブート	11
コマンド・ライン	39
コンソール・アウトプット	11
コンソール・インプット	11
コンソール出力	48

コンソール・ステータス	11
コンソール入力	47
コンソール・バッファ	66
コントロール・ライン	305

サ

ザイログ型式	186
シーケンシャル・ファイル	37
シーケンシャル・リード	121
システム・コール	34, 47, 51
純LISP	270
シンボル・テーブル	178, 183, 190
スキュー	13, 19
スキュー・ファクタ	4, 18
スクラッチ・エリア	39
スクラッチ・パッド・エリア	3
スクリーン・エディタ	294
スタック	49
ステータス・チェック・ルーチン	14
セクタ・トランスレータ	13
セクタ・トランスレータ・テーブル	16
セクタ・トランスレータ・ベクトル	17
セットDMAアドレス	13
セット・セクタ	12
セット・トラック	12
セル	287
セレクト・ディスク・ドライブ	12

タ

ターミナル	301
チェックサム	308
中間コード・ファイル	227
ディスク・パラメータ・アドレス	114
ディスク・パラメータ・テーブル	15
ディスク・パラメータ・ブロック	18, 19, 114
ディスク・パラメータ・ヘッダ	6, 16
ディスク・ライト	13
ディスク・リード	13
ディスプレイ・ウィンドウ	287
ディレクトリ	34
ディレクトリ・エントリ	77
ディレクトリ・コード	81
データ・ブロック	28
デバイス・セレクト・ルーチン	14
デブロッキング	30
電子早見帳	178
トランスレータ	207

ナ

ニーモック	165
-------	-----

ハ

バージョンNo.	72
ハンチ・アウトプット	12
ファイル・アトリビュート	111
ファイル・コントロール・ブロック	34
ファイル・サイズ	139
フィジカル・セクタ	19, 30
フィジカル・デバイス	61
ブロッキング	30, 32
プロトコル	308
ホームへのシーク	12
ボーレート	305
ホスト・バッファ	31

マ

マクロ・アセンブラ	165
マクロ・コール	169
マクロ・ライブラリ	169, 187
モジュール	185
モジュール別ソフト開発法	185

ヤ

ユーザー・エリア	118
ユーザー・コード	118

ラ

ライト・プロテクト	106
ランタイム・システム	227
ランタイム・ライブラリ	201
ランダム・ファイル	37
ランダム・リード	122
ランダム・レコード	140
リーダ・インプット	12
リスト・アウトプット	12
リスト・ステータス	13
リロケータブル	165
リロケータブル・オブジェクト	181, 183
リンク	181, 183
リンク・ローダ	193
ログイン・ディスク	75
ログイン・ベクトル	97
ロジカル・セクタ	18, 30
ロジカル・デバイス	14, 61

ワ

ワークシート	287
--------	-----

応用CP/M

アスキー・ラーニングシステム③実習コース

1982年12月20日 第1版2刷発行

定価1,800円

著者 村瀬 康治

発行者 塚本慶一郎

発行所 株式会社 **アスキー**

〒107 港区南青山5-11-5 住友青山ビル5F

振替 東京7-57496

電話 03-486-7111(代表)

©1982 ASCII Corporation. Printed in Japan.

本書は著作権法上の保護を受けています。本書の一部あるいは全部について（ソフトウェア及びプログラムを含む）、株式会社アスキーから文書による許諾を得ずに、いかなる方法においても無断で複写、複製することは禁じられています。

編集担当者 井芹昌信

印刷 壮光舎印刷

ISBN4-87148-602-8 C3055 ¥1800E

貸出期限票

キハラ No. 1452

A
3

22888



アスキー出版局

定価1,800円

ISBN4-87148-602-8 C3055 ¥1800E